



2023

Code Beats - Teaching Computer Programming Coding via Hip Hop Beats

Douglas Lusa Krug
Virginia Commonwealth University

Follow this and additional works at: <https://scholarscompass.vcu.edu/etd>

© The Author

Downloaded from

<https://scholarscompass.vcu.edu/etd/7288>

This Dissertation is brought to you for free and open access by the Graduate School at VCU Scholars Compass. It has been accepted for inclusion in Theses and Dissertations by an authorized administrator of VCU Scholars Compass. For more information, please contact libcompass@vcu.edu.

CODE BEATS - TEACHING COMPUTER PROGRAMMING CODING VIA HIP
HOP BEATS

A dissertation submitted in partial fulfillment of the requirements for the degree of
Doctor of Philosophy at Virginia Commonwealth University.

by
DOUGLAS LUSA KRUG
Ph.D. Candidate

Director: Kostadin Damevski,
Associate Professor, Department of Computer Science
Virginia Commonwealth University

Virginia Commonwealth University
Richmond, Virginia
May, 2023

Acknowledgements

It would not be possible to go through this journey without the unconditional support that I received from my family. Their encouragement and understanding have been crucial to me since the moment I decided to start the whole Ph.D. process. So, I dedicate this work to my wife, Nariel, my daughter Brenda, my son Luiz, and my parents, Lauro and Marileusa.

During my Ph.D. journey, I had the chance to be advised by Dr. David Shepherd, who understood my strengths and weakness and made me use them toward my goals. Thanks, Dave!

Also, during the last mile, Dr. Kostadin Damevski facilitated my path until I could cross the finish line. Thanks, Kosta!

Finally, this would not be possible without God. Everything at His time. So, I'm grateful for having this experience.

Table of Contents

Table of Contents	iii
List of Tables	vi
List of Figures	vii
Abstract	x
1 Introduction	1
1.1 Research Goals	3
1.2 Dissertation Structure	5
2 Background and Literature Review	6
2.1 Disparities in Access to Computer Science	6
2.1.1 Structural Barriers	7
2.1.2 Perceptions of Programming	9
2.2 Using Music to Teach Coding	10
2.2.1 Platforms to Teach Coding using Music	11
2.2.2 Experiments Using Music to Teach Coding	12
2.3 Culturally Relevant Pedagogy	14
2.4 Discussion	15
2.4.1 Dissertation Contributions	16
3 The Code Beats Approach	17
3.1 Curriculum: Music + Computer Science Concepts	17
3.2 Genre Choice	24
3.3 Software Tooling	25
3.4 Camp Format	27
3.4.1 Curriculum Distribution	28
3.4.2 Core Segments	29
3.5 Activities	31
3.5.1 Example Activity in TunePad.live	33
3.5.2 Example Activity in TunePad.com	38
3.5.3 List of Activities	40

4	Measuring Engagement Toward Computer Science	42
4.1	Research Questions	43
4.2	Methods	44
4.2.1	Participant Demographics	44
4.2.2	Data Collection	44
4.2.3	Data Analysis	45
4.3	Results	46
4.3.1	RQ1: Student Engagement	46
4.3.2	RQ2: Developing Creators	50
4.4	Limitation and Threats to Validity	51
4.5	Conclusions	51
5	Indirectly Engaging Adult Learners	53
5.1	Related Work	54
5.2	Research Question	56
5.3	Methods	56
5.3.1	Study Design and Data	56
5.3.2	Participant Demographics	57
5.3.3	Data Analysis	58
5.4	Results	59
5.5	Discussion	67
5.5.1	Perception about Computer Programming - Before <i>Code Beats</i>	67
5.5.2	Perception about Computer Programming - After <i>Code Beats</i>	67
5.6	Limitations and Threats to Validity	68
5.7	Conclusion	68
6	Engaging Students with No Musical Background	70
6.1	Context of Work	71
6.1.1	Computer Science Education in Brazil	71
6.1.2	Music Education in Brazil	72
6.2	Adapting <i>Code Beats</i> for Context	73
6.3	Research Question	75
6.4	Methods	76
6.4.1	Study Design	76
6.4.2	Participant Demographics	76
6.4.3	Data Collection and Analysis	77
6.5	Results	78
6.5.1	Students' Motivation	78

6.5.2	Direct Observation	82
6.6	Limitation and Threats to Validity	84
6.7	Conclusion and Future Work	84
7	Study of Scaffold-based Activities for Music Coding	86
7.1	Related Work	87
7.1.1	Scaffold-Based Curricula and Activities	87
7.1.2	Scaffolding Music-based Programming	88
7.2	Adapting Scaffolding for Music Coding	89
7.2.1	Complete the Code	89
7.2.2	Buggy Code	90
7.2.3	Reorder the Code	91
7.3	Research Questions	92
7.4	Methods	93
7.4.1	Participant Demographics	93
7.4.2	Study Design	94
7.4.3	Data Analysis	95
7.5	Results	96
7.5.1	RQ1: Difficulty	96
7.5.2	RQ2: Correctness	98
7.6	Limitation and Threats to Validity	101
7.7	Conclusion	102
8	Using Domain-Specific, Immediate Feedback in <i>Code Beats</i> to Support Students	103
8.1	Related Work	104
8.1.1	Feedback to Improve Code Learning	104
8.2	Background	106
8.2.1	Domain-Specific Immediate Feedback	106
8.3	Research Questions	110
8.4	Methods	110
8.4.1	Study Design and Data	110
8.4.2	Participant Demographics	112
8.4.3	Data Analysis	112
8.5	Results	113
8.5.1	RQ1: Correctness	113
8.5.2	RQ2: Students Perception on Feedback	114
8.6	Discussion and Conclusion	116

8.7	Limitation and Threats to Validity	118
9	Conclusion	120
9.1	Research Contributions	120
9.2	Significant Findings	122
9.3	Future Work	124
	References	126
	Appendix A List of Activities - TunePad.live	145
	Appendix B List of Activities - TunePad.com	148

List of Tables

1	Curriculum Distribution - Final Version	29
2	Coding with Doug - Videos	30
3	Decoding the Beat - Videos	30
4	Vocabulary Definitions - Videos	31
5	Differences between pre and post-surveys	46
6	Curriculum Distribution - Brazilian Edition	74
7	List Code Beats of Activities - Brazilian Edition	75
8	Pre and Post-course Survey	79
9	Programming Concepts and Activity Types	94
10	Code Beats Camps - Number of Participants	124
11	List with Code Beats Activities 1	146
12	List with Code Beats Activities 2	149

List of Figures

1	Code Example - Modularization	23
2	Example - Parallelism	24
3	Code Example in TunePad.com	26
4	Code Example in TunePad.com - Drums	27
5	In-Class Activity - Example - Instructions	33
6	In-Class Activity - Example - Melody	34
7	In-Class Activity - Example - Bass	35
8	In-Class Activity - Example - Drums (Hi-hat)	36
9	In-Class Activity - Example - Drums (Snare)	36
10	In-Class Activity - Example - Drums (Kick)	37
11	In-Class Activity - Example - Timeline	38
12	In-Class Activity - Example 2 - Instructions	38
13	In-Class Activity - Example 2 - Videos	39
14	In-Class Activity - Example 2 - Melody	40
15	Survey Results: statements with a positive view of computer science . . .	48
16	Survey Results: statements with a negative view of computer science . . .	49
17	Number of original beats per student over time	50
18	Q1 - What did you think before?	59
19	Q2 - What do you think now?	61
20	Q3 - Interested in learning more?	63

21	Q4 - Interested in a career?	64
22	Q5 - Did the use of music change your attitude?	65
23	Difficulty - Results for Complete the Code and Buggy Code	97
24	Difficulty - Results for Complete the Code and Reorder the Code	98
25	Correctness - Results for Complete the Code and Buggy Code	99
26	Correctness - Results for Complete the Code and Reorder the Code	100
27	Example of Hints	107
28	Example of Feedback - 1	108
29	Example of Feedback - 2	109
30	Percentage of Correct Solutions	114
31	Students Motivation - Feedback System	115

Abstract

Computer programming is a crucial skill for future professionals, not only those working as computer programmers but most modern workers. To train the next generation, society has created many initiatives to introduce computer programming to young students. These initiatives range from classes in a formal academic setting to informal, extracurricular sessions in after-school and summer camps. Even with the increasing offer of these initiatives to expand the opportunities to learn computer programming, the interest in computer programming remains low, especially among populations underrepresented in computing.

This lack of interest could be impacted by stereotypical views of computer programming as tedious and difficult to learn and the idea that programming is only for those with a “geek” gene. Therefore, introducing students to a different side of computer programming, such as its ability to make high-quality music in connection to the use of culturally relevant pedagogy, may be an essential tool in changing students’ perceptions of this field.

To investigate this, this dissertation describes and evaluates my approach that introduces the foundational concepts of computer programming using music. First, it investigates prior work that has used music to teach programming. Next, it describes my approach and curriculum design, which combines programming with hip-hop music. Then, it analyzes my approach’s impact on attracting and engaging students in several contexts. Finally, it demonstrates how pedagogical approaches commonly used in computer science education can be adapted to this musical context without losing effectiveness. The results indicate that my approach attracts, motivates, and

engages students in computer science, a promising step in the effort to broaden the appeal of computer science to increase diversity.

CHAPTER 1

INTRODUCTION

Computer programming is considered a crucial skill for future professionals, not only for those who are interested in working in the area but also for those who look for positions that are not directly related to computer programming or in the Computer Science (CS) field, as many jobs will require basic knowledge of computer programming [1]. Nowadays, thousands of positions are open for professionals skilled in computer programming, with the forecast of thousands more to be created in the short term, increasing the already existing shortage of professionals for this field [2].

To train the next generation of professionals in these skills, more and more initiatives are being created to teach computer programming to young students. These initiatives include specific classes in a high school curriculum, summer camps, community coding events, and guided online curricula. Even with the increased offering of computer programming courses, interest remains low, especially among populations underrepresented in computing [3, 4]. This lack of interest could be impacted by stereotypical views of computer programming as boring or difficult to learn [5, 6]. Students may also believe that programming is only for socially awkward people with no other interests but technology [7]. Combating these perceptions is essential, as they lead many students to ignore or avoid opportunities to engage with this vital skill.

Introducing students to a different side of computer programming, such as its ability to make high-quality music, may be an essential tool in changing students' perceptions of this field. In particular, middle school is the ideal time to try and change

these perceptions; in middle school, students typically undergo intense physical and physiological development, which is crucial to their intellectual and behavioral development [8]. For instance, it has been shown that active participation in out-of-school time during this period in a child's life predicts subsequent values and self-conception of their abilities [9]. If we can reach students during this time, we may be able to build the confidence and competence necessary to promote a lifelong engagement with computer programming, even if it is not their primary occupation.

One promising approach to engaging all students in CS is using culturally relevant pedagogy [10, 11]. Another practical approach is to leverage students' existing interests, such as music [12]. *Code Beats*, an innovative and novel approach where this dissertation is situated, uses both, leveraging hip-hop music to capture the imaginations of a wide array of urban youth of color from diverse places of origin [13]. The use of hip-hop aims to overcome structural and social barriers to access to CS classes. The *Code Beats* approach uses a vernacular culture, incorporating a musical genre that is part of our targeted audience, aiming to, in the first step, attract the students' attention and, in the second step, keep them motivated to learn computer programming using something that is from their interest.

My thesis statement is: *introducing students to computer programming using high-quality, culturally relevant music, will improve students' perceptions of the computer science field.* To investigate this thesis, the primary goal was to design and analyze a curriculum that uses hip-hop beats to teach the foundational concepts of computer programming to attract and engage students in the CS field. I started focusing on students currently in middle school, understanding that they are in an essential phase of their lives to make choices for their future. Then, the range of the population increased, experimenting with the approach with adult learners, who, in the scope of this dissertation, are those adults looking to improve their skills through

education, either to improve their current job or to seek better opportunities.

Additionally, the approach was tested with a population without previous music knowledge. Finally, although the musical genre choice was motivated to broaden the diversity in computer science, the participants are not limited to those from underrepresented groups in CS.

1.1 Research Goals

Analyze the current use of music to teach computer programming. The use of music to teach programming has increased in the past years as an alternative to attract and engage students in CS. As a result, multiple platforms were explicitly created to apply the computer programming constructs in the music composing process. My work does not intend to create a new platform, as I understand it is possible to use the existing platforms to achieve the primary research goal. Nevertheless, understanding the platforms, their strengths, and their weakness is crucial for this work to choose the platform that best fits this research’s goal. Understanding the existing curricula is also essential to building and validating my approach.

To accomplish this goal, I searched for, analyzed, and experimented with the computer programming platforms that have published research results that use music to attract and engage students toward CS. This investigation led to the choice of the platforms to create and apply the approach.

Design a curriculum that integrates computer programming concepts with music. The curriculum is an essential piece of every kind of instruction. It must guarantee that the main topics are addressed in the proper order and with sufficient depth. When using a novel approach to teach computer programming, it is vital to understand how to build the connections between the computer programming constructs and the domain topic, music, in this work’s scope. Using a natural connection

between topics is extremely important to keep it engaging and not force unrelated concepts.

To put my approach into practice, I designed a curriculum with the foundational computer programming concepts mapped to the musical concepts, mainly from hip-hop.

Analyze the impact of this approach on student engagement in computer science. The approach presented in this dissertation does not intend to teach the students all the foundational computer programming concepts deeply. Instead, its main objective is to attract and engage students in computer science. Thus, assessing this approach regarding engagement and motivation in the CS field is necessary to evaluate its effectiveness.

To analyze the impact of this approach, serving as a basis for the community, I assessed the students' impressions and knowledge before, during, and after exposure to *Code Beats* approach, using surveys, focus groups, and students' artifacts.

Investigate the use of educational techniques to facilitate the students' experience. Naturally, the first contact that someone has with something new can create friction, and if this friction is not managed, it can develop problems such as a lack of confidence and self-created barriers. Understanding the pain points of students during the activities is essential to create ways to overcome those problems.

Aiming for a better experience for students during the activities, I analyzed how students struggled during music-related programming activities, identifying common issues. Once identified, I applied education techniques, such as scaffolding and immediate feedback, to reduce the friction of learning and improve students' experience.

1.2 Dissertation Structure

This dissertation is organized as follows: Chapter 2 provides a background and literature review of the dissertation topics; Chapter 3 explains my approach, *Code Beats*, the software tooling choice, the curriculum, and the camp format; Chapter 4 presents first use of *Code Beats*, reporting the results of the first camp; Chapter 5 reports using *Code Beats* with adult learners; Chapter 6 reports using this approach with a population without previous experience with music; Chapter 7 presents the use of different scaffold-based activity types to measure the difficulty and correctness across the selected activity types; Chapter 8 reports the use of a domain-specific, immediate feedback system to guide students during the activities-solving process; Finally, Chapter 9 discusses the overall results and conclusions and future work.

CHAPTER 2

BACKGROUND AND LITERATURE REVIEW

This chapter presents the literature, providing background to situate the work. It starts by stating the current barriers to broadening participation in computer science. Next, it discusses the related work that uses music to teach coding. Then, it situates the work in the use of culturally relevant pedagogy. Finally, it ends with a discussion about the existing gaps in current work.

2.1 Disparities in Access to Computer Science

Research examining race, ethnicity, and gender disparities in access, preparation, and persistence in the CS field has shown barriers that are convergent [3]. These barriers can be divided into two categories which we briefly describe and then discuss in detail:

Structural barriers: disparities in access to rigorous computer science work; lack of engaging and relevant computing curriculum; lack of diverse role models and peer networks in computer science; and implicit bias affecting recruitment, hiring, and promotion in technology workplaces.

Social and psychological barriers: misconceptions about the field of computing; perceptions of computer science as lacking cultural and social relevance; stereotype threat associated with being a member of a marginalized group; and stereotypical cues in the classroom and workplace environment.

2.1.1 Structural Barriers

The disparity by race and ethnicity is evident; the overall population in the 2020 U.S. Census [14] shows that the general population is formed by 57.8% of White, 12.1% of Black or African-American, and 18.7% of Hispanic or Latino. Simultaneously, according to the labor force characteristics report [15], the population employed in computer and mathematical occupations (total of 5,352,000 positions) is represented by 65.7% of White, 8.7% of Black or African-American, and 7.8% of Hispanic or Latino.

One way to increase diversity in CS is through formal education, such as higher education courses. Advanced Placement tests, like AP Computer Science A ¹ and AP Computer Science Principles ², offer college credits equivalency for those that take the exam and perform above a certain threshold. Offering regular CS classes in High Schools encourages students to take the AP Computer Science A or AP Computer Science Principles tests.

Currently, according to the 2020 State of Computer Science Education report [16], only 47% of High Schools in the U.S. offer CS courses for their students. The overall demographics of high school students are formed by 47% white students, 15% African-American students, and 27% of Hispanic or Latino students. If one examines only the high schools where AP CS is being taught, the demographics are quite similar: 48% white, 13% African-American, and 26% of Hispanic or Latino.

The problem is in the number of students taking the AP CS exams. The demographic is formed by: 44% of white students, 6% of African-American students, and 17% of Hispanic or Latino students [16]. It shows that African-Americans and

¹<https://apstudents.collegeboard.org/courses/ap-computer-science-a>

²<https://apcentral.collegeboard.org/courses/ap-computer-science-principles>

Hispanic or Latino students are less likely to take the AP computer science exam, even when they attend a school that offers CS courses.

Noteworthy, the proportion of students who take the AP CS exams from each demographic has increased. For instance, the percentage of African-American students who took the AP CS exam in 2014 was 3.9%, growing to 5.7% in 2019. For the Hispanic or Latino population, the difference is higher. In 2014, the percentage was 8.8%, and in 2019 it increased to 16.6%. This increment in participation coincides with the introduction of AP Computer Science Principles [16], which rather than the AP Computer Science A, introduces students to the foundational concepts of the field.

Each state has its policies to broaden students' participation in CS, adopt standards, and create state plans at the state level. However, it is also true that each state has its demographics. For instance, the overall high school student demographics of the Commonwealth of Virginia is formed by 48% of white students, 22% of African-American students, and 16% of Hispanic or Latino students. The demographics in high schools that offer CS courses are also similar: 49% of white students, 19% of African-American students, and 17% of Hispanic or Latino students [16].

Statewide, as nationwide, the disparity can be observed in the number of students taking the AP CS exams, where the demographics are formed by: 44% of white students, 8% of African-American students, and 8% of Hispanic or Latino students [16]. So, in the Commonwealth of Virginia, African-American and Hispanic or Latino students are each three times less likely than their White peers to take an AP CS exam when they attend a school that offers it.



The proportion of the population studying or employed in CS differs from the overall population in the U.S., with African-Americans and Hispanics or Latinos being underrepresented in the area.

2.1.2 Perceptions of Programming

In addition to offering computer programming classes in high schools, several initiatives use informal learning environments to teach foundational skills in computer programming. For example, they go from unplugged activities [17] to the use of robots [18] to the use of block-based environments [19, 20, 21] and video-games development [22].

Even with this variety of ways to introduce computer programming and CS concepts, some stereotypes of people studying or working with CS persist. For instance, some people think that code is too difficult to learn, that coding is boring, and that it is a solitary occupation that does not generalize to other fields of interest [1]. Furthermore, some people think that to study or work in CS, one must have a natural computing ability, creating beliefs of a requirement of a “geek gene” [23]. Moreover, some characteristics were identified as a requirement to “fitting in CS”: singularly focused on CS; asocial; competitive; and male [24]. These stereotypes can create negative attitudes toward CS, discouraging individuals from learning computer programming [1] or just presenting CS as an unappealing field [23].

Those “stereotypes” of computer scientists are pervasive, and the system of inequality in CS appears to rely on stereotypes about the practitioners [23]. However, it is possible to refer to the term “stereotypes” as “narratives”. They are connected but conceptually distinct. The term “narratives” captures the notion that beliefs about computing and computer scientists are communicated by and between people and therefore can and do change [23]. Changing the narratives makes it possible to

change the thoughts about computing and computer scientists.

In addition to the stereotypes about one who studies or works in the CS field, learning or working with computer programming is perceived as difficult. Historically, learning to program is perceived as difficult for many people. Furthermore, introductory programming courses typically have high student dropout and failure rates [25]. The claim that introductory programming courses have high dropout and failure rates has been one of the main practical concerns for computing teachers and computing education researchers [25]. However, studies that report those rates fail to compare with other courses in the same institution or across different institutions [25].

This narrative created by the CS stereotype and the difficulty of computer programming is one of the reasons for the disparities by race, ethnicity, and gender in access, preparation, and persistence in CS courses and workforce still exists, with women, African-Americans, and Latinos participating at rates lower than their representation in the population [3].



A stereotype about computer science, mainly computer programming, still exists, creating a belief that it is boring and difficult to learn, requiring a natural ability to succeed.

2.2 Using Music to Teach Coding

One way to attack the barriers mentioned above is by using different ways to present and teach computer coding. Among these ways that is starting to show promise is using music to teach coding. Several groups have used music intending to broaden participation in CS, motivating students to learn to program. This section will first present the platforms and their main features. And next, the most relevant works that use music to teach coding, along with their main characteristics.

2.2.1 Platforms to Teach Coding using Music

EarSketch [26], a programming environment for remixing music developed at Georgia Tech, has been shown to increase engagement for all, especially for both females and racial minorities. The EarSketch approach “..focuses on the level of beats, loops, and effects more than individual notes, enabling students with no background in music theory to begin creating ..music immediately, focusing on higher-level musical concepts such as ..mixing.” [26]. That is, EarSketch emphasizes immediacy.

While EarSketch favors immediacy, other platforms emphasize depth. For example, JythonMusic [12] allows users to generate individual notes or chords instead of just mixing existing audio files. However, JythonMusic has been used in the context of a first-year university music appreciation course at the College of Charleston, where many genres of music are discussed, and a deep knowledge of music theory is required by its current curriculum [12].

Sonic Pi [27] is a coding platform initially created to run on a Raspberry Pi³ that had as its objective “...create a minimal domain-specific language (DSL) that would a) provide a musically engaging experience for students; and b) demonstrate a reasonably wide range of basic computational concepts.”[27]. Sonic Pi works as a text-based IDE and allows the use of individual notes and additional samples, such as a hi-hat or a guitar.

TunePad [28] is another approach to engaging students via music mixing, created in a computational notebook style. Its mix of visualization and computation, focusing on usability and ease of use, makes it well-suited for a broader audience. Using TunePad, it is also possible to use individual notes to “play” pitched instruments or use a set of drum sounds. Furthermore, in TunePad, one can write its code and

³<https://www.raspberrypi.org/>

see the distribution of its music in an instrument timeline. Thus, in this dissertation research, we chose TunePad as the platform.

2.2.2 Experiments Using Music to Teach Coding

Besides the works that propose platforms that integrate coding with music, some researchers also compare students’ engagement, motivation, future in CS, and learning when using these environments. In some cases, the paper presenting the platform also presents the initial results. For instance, the paper that presented EarSketch [26] also reported its curriculum’s first idea and results. The curriculum was designed for use in a five-day summer workshop setting and was based on the CS principle topics within the context of loop-based composition, targeting high school students. The results from this initial experiment suggest that students’ attitudes positively and statistically increased in “Computing Confidence”, “Motivation to Succeed in Computing” and “Creativity”.

Another paper briefly describes a curriculum, where music technology and computing concepts were taught together, always linking new computational concepts to musical applications. The results of a pilot experiment with high school students suggest that EarSketch’s music and computing learning environment effectively teaches introductory computing concepts in a formal academic course, also improving students’ attitudes toward CS [29]. Finally, another experiment using EarSketch with high school participants, where the curriculum focuses on broader views of CS, such as computational thinking, creativity, abstraction, programming, and the Internet, provided evidence showing significant increases in intent to persist and content knowledge in computing [30].

The paper introducing Sonic Pi also reports preliminary findings of a pilot study. This study focused on middle school students and had five one-hour lessons. The

paper did not provide more details about the curriculum and lessons and did not use surveys or other instruments to measure students’ perceptions or knowledge. However, through the researcher’s observations, the paper’s findings show that all students successfully acquired basic competence in programming [27].

It is also possible to find experiments performed by researchers different from those that proposed the platforms. For instance, two experiments using Sonic Pi conducted by researchers unrelated to the Sonic Pi platform proposal are reported next. The first report results from workshops held for high school students [31]. This paper does not report curriculum or activities details but claims that basic programming concepts were introduced and successfully applied by the participants. The second reports an experiment with middle school students, reporting growth in programming attitudes [32]. This work reports activity plans and concepts taught, but does not use a specific music genre.

The results of preliminary tests using TunePad with middle school students are included in the paper where TunePad was first described [28]. It reports that students found TunePad engaging, making it a potential platform for introducing learners to computational thinking skills. In another experiment with middle school students, where TunePad was part of more extensive programmatic activities, such as learning about the history of hip-hop, making and tinkering activities, and work of “real” scientists, participants showed significant attitudinal gains in interest, self-confidence, enjoyment, and intention to persist in CS [33]. Finally, there is a report on using EarSketch and TunePad at the same camp, where findings suggest that using different approaches may help engage students with varying levels of music and coding experience [34].



Using music to teach coding is being shown as promising, with exciting results in attracting and engaging students in computer programming.

2.3 Culturally Relevant Pedagogy

Culturally responsive computing in education is a relatively new approach with the goal of potentially increasing student interest from underrepresented ethnic groups [35]. Culturally responsive education is not limited to raising test scores in some subjects. Instead, it can improve the inclusive scope to better serve a multicultural society's needs, inclusively inspiring a new generation of computing professionals.

In terms of culturally relevant education, six themes or strategies are commonly implemented in K-12 [36]:

Sociopolitical consciousness raising: defined as a practice in computing that allows students to reflect and build awareness of current social-political issues in the world.

Heritage culture through artifacts: described as a culture that is transferred through ancestral roots.

Vernacular culture: encompasses the local social environment of participants, incorporating contemporary cultural practices relevant to the participants.

Lived experiences: involves the recognition of students' lived experiences, connecting to the participants' real-world context and the participants' self-identity.

Community connections: uses a strategy of building community connections involving community members, students, and teachers as brokers of cultural knowledge in computer programming.

Personalization: that focuses on product customization in student-centered projects.

Unlike the heritage culture, vernacular culture corresponds to domains such as rap music and a wide variety of other popular activities that children from underrep-

resented groups feel some sense of ownership [35].



Incorporating contemporary cultural practices relevant to participants can increase their interest in a specific topic, such as computer science.

2.4 Discussion

The CS field is an important field in terms of open positions to be filled, but it is a field that lacks diversity. One reason for that is the structural, social, and psychological barriers. These complex and interconnected barriers create and perpetuate significant disparities in the computer science field where, even with all initiatives already in place, there is still work to be done.

Informal learning in CS is a fundamental approach to attracting students to the field and letting them explore the opportunities. Unfortunately, most informal learning opportunities attract students already interested in CS. Therefore, narratives that stereotype CS people may be problematic in attracting one not yet interested in this area.

In a more formal environment, such as in high schools, CS courses are offered, but the diversity of students taking these courses is still a problem. One may argue that if the student does not experience computer programming, how would the student opt to choose a year-long class about computer programming? It may be a broad impact because if one does not see computer programming as something doable and enjoyable during middle school, one may not choose CS as an elective for high school.

The use of culturally relevant pedagogy in CS education sounds promising in modifying the narrative about CS and, at least, attracting students that are not yet interested in CS to “give it a chance”. One way to do that is to use vernacular culture, incorporating contemporary cultural practices relevant to the participants, such as

hip-hop for African-Americans and Hispanic or Latino Americans.

Music programming has shown promise, with exciting results in improving the aptitudes towards CS and being effective in teaching the foundational concepts of computer programming. However, most studies presented do not fully integrate the content with a specific musical genre to attract the population underrepresented in the field. Furthermore, the studies are focused on a specific population (i.e., high school students), not being tested with multiple populations, with variations of age, culture, prior coding experience, and prior music experience.

2.4.1 Dissertation Contributions

This dissertation contributes to the CS community by presenting an approach that incorporates culturally relevant aspects to teach computer programming using music. It aims to broaden CS education by attracting those who the historically traditional approaches would not attract.

Additionally, it contributes to the growing body of studies that use music to teach coding testing it with different populations, such as adult learners, in different cultures and contexts, and people without a music background. Finally, it describes and tests the use of educational techniques, such as scaffolding and task feedback in the context of music, providing guidance on how to use it and what works better.

CHAPTER 3

THE CODE BEATS APPROACH

This dissertation presents *Code Beats*, an innovative approach that uses music, specifically hip-hop, to teach the foundational computer programming concepts while creating beats. *Code Beats* was proposed by a group of researchers that I worked with, to develop new ways of broadening the participation of underrepresented populations in CS and attracting more and more people to this field. For that, actual hip-hop songs are used, transcribed to a coding platform, as background for the activities, using them as a means to complete a coding task that connects to a musical concept. This chapter presents the computer programming concepts taught, the motivation for the genre choice, the software tooling used in my approach, and the camp format, with its segments and activities.

3.1 Curriculum: Music + Computer Science Concepts

With the *Code Beats* curriculum, it is possible to teach various computer programming concepts, including sequencing, variables, constants, functions and parameters, lists, repetitions, modularization, and parallelism. This list covers almost all concepts from Units 4, 5, and 7 in the code.org CS Principles Curriculum Guide [37].

The main point of using music as a background for teaching computer programming is the relation that both fields have, allowing for smoothly connecting the concepts in an almost natural way, as follows:

Sequencing: This is one of the first concepts introduced in our class. It might be natural for those more experienced in computer programming. Still, it is imperative

to teach beginners how a computer will read and perform the commands, line-by-line, in order, from top to bottom, and, consequently, how it respects the order that a program is written. This concept can be naturally connected with music. Making an analogy, a musician will follow a sequence of musical notes to compose or play a piece of music.

For instance, at Listing 3.1, we have a code extract from a melody, where the order or musical notes, represented by computer commands, matters. So, suppose we change the order of any of the commands (function *playNote*) or change the order of the parameters (MIDI number). In that case, the melody will not be consistent with the original song playing in the background.

Listing 3.1: Code Example - Sequencing

```
1 playNote(49)
2 playNote(56)
3 playNote(57)
4 playNote(56)
```

Variables and constants: The concept of variables can be hard to teach, mainly if you cannot link it to any concrete example. Our approach can almost naturally solve this problem by giving a name (musical note) to a MIDI number. One can say that it can be characterized as a constant, and in essence, it is, but it can be transformed into a variable if we use the same musical note and change the octave, for instance, increasing the current MIDI number by 12. One natural aspect related to music is the name of musical notes, where each pitch receives a name, in this case, a musical note name.

At Listing 3.2, we have one example of the use of constants, where we have the same melody from Listing 3.1 but now using the names of the musical notes as parameters through the use of constants.

Listing 3.2: Code Example - Constants

```
1 Cs = 49
2 Gs = 56
3 A  = 57
4
5 playNote(Cs)
6 playNote(Gs)
7 playNote(A)
8 playNote(Gs)
```

One example of using variables can be seen at Listing 3.3, where the musical note “F” is used in two different octaves in this piece of code. Initially, its value is 53 (line 2), which is F in the 3rd octave, then it changes to 65 (line 14), adding 12, which is F in the 4th octave, and later comes back to the 3rd octave (line 17), subtracting 12.

Listing 3.3: Code Example - Variables

```
1 Eb = 51
2 F = 53
3 G = 55
4 Ab = 56
5 C = 60
6 D = 62
7 vol = 120
8 vol2 = 75
9
10 playNote(Eb, beats = 0, sustain = 4, velocity = vol2)
11 playNote(C, beats = 0, sustain = 4, velocity = vol2)
12 playNote(G, beats = 2, sustain = 4, velocity = vol2)
13 playNote(D, beats = 1, velocity = vol)
14 F = F + 12
15 playNote(F, beats = 1, velocity = vol)
16
17 F = F - 12
18 playNote(F, beats = 0, sustain = 4, velocity = vol2)
19 playNote(Ab, beats = 0, sustain = 4, velocity = vol2)
20 playNote(C, beats = 2, sustain = 4, velocity = vol2)
21 playNote(D, beats = 2, velocity = vol)
```

Functions Calls and Parameters: As the most important commands used in our class are functions, the parameter is a crucial concept to teach. For instance, we

use parameters to play a specific musical note in functions *play* and *playNote* where the musical note (a MIDI number or a constant) works as a value for a parameter. Additionally, we can control other behaviors with parameters, such as how long a note will be played. This is a perfect example of the use of parameters in music, where the musician must read this “parameter” and execute the “function” (playing an instrument) for that duration of time.

At Listing 3.4, we have examples of functions *rest* and *playNote* and their parameters. At first, we use only the parameters *beats* that control for how long the pause will be. At second, we use two parameters, *note* and *beats*. The first can receive a number representing a pitch value or a drum sound, and the second one controls how long the sound will play.

Additional parameters can also be used in function *playNote*. For instance, at Listing 3.3, lines 10 to 13, we have the parameters *sustain* and *velocity*. The first allows the note to play for time longer than the value used for parameter beats. The second one says how loud the note will sound.

Listing 3.4: Code Example - Functions and Parameters

```
1  clap = 58
2
3  rest(2)
4  playNote(clap , beats = 1)
```

Lists: Lists (data structures) can be seen as a complicated concept to teach to beginners in computer programming. Even the concepts of constant and variable may be difficult, so imagine explaining a constant where you can store and use more than one value simultaneously. Linking to a concrete example makes explaining abstract constructs, such as data structures, easier to explain. In our approach, we are using lists and limiting the use of a list to store multiple values, use all values simultaneously, and use each one individually. Connecting lists with music makes teaching it a little

easier because we use the musical concept of “chords” where a group of harmonic musical notes is played simultaneously, and “scale” where you can store the musical notes from a musical scale.

At Listing 3.5, we have one example of the use of lists where we are creating lists - chords (lines 4, 8, and 12) with multiple variables - musical notes. So when we “play” a chord (lines 14 to 18), we are “playing” all musical notes simultaneously.

Listing 3.5: Code Example - Lists

```
1 Ab = 56
2 Db = 61
3 E = 64
4 Db_min = [ Db, E, Ab ]
5
6 A = 57
7 Gb = 54
8 Gb_min = [ Gb, A, Db ]
9
10 Eb = 63
11 B = 59
12 Ab_min = [ Ab, B, Eb ]
13
14 playNote(Db_min, beats = 3)
15 playNote(Gb_min, beats = 5)
16 playNote(Db_min, beats = 3)
17 playNote(Gb_min, beats = 3)
18 playNote(Ab_min, beats = 2)
```

Repetitions: Repeating a command or a group of commands is a very frequent task when we write “regular” programs (e.g., visit elements in a list). In our approach, we are teaching two different types of repetition; one is numerically controlled (i.e., *for i in range(0, 4):*), and another one is linked to a list (i.e., *for n in notes:*). In fact, we have automatic repetitions for both tools that we used (Sonic Pi and TunePad). With Sonic Pi the commands must be written inside a *live_loop*, which means that this specific group of commands will repeat until the user asks to stop the program. In TunePad each track is structured to repeat for (at least) 4 beats when we use

individual tracks. Relating to music, we can observe repetitions in a drum sequence that repeats a hi-hat several times, and also a melody that repeats a group of notes during the song.

For instance, at Listing 3.6, we have a hi-hat track that is playing the hi-hat eight times, controlled by a numeric repetition (line 3), each time for 0.5 beats (line 4).

Listing 3.6: Code Example - Repetitions - Numeric controlled

```
1 hi_hat = 4
2
3 for i in range(0, 8):
4     playNote(hi_hat, beats=0.5)
```

At Listing 3.7, we are playing a melody using the order of the notes (line 7) retrieved from a list (line 5). As all the notes are being played for the same amount of time, it is possible to use a repetition controlled by lists.

Listing 3.7: Code Example - Repetitions with list

```
1 D = 74
2 C = 72
3 B = 71
4
5 notes = [ D, C, B, B ]
6
7 for note in notes:
8     playNote(note, beats = 2)
```

Modularization: This is another concept that might be hard to teach to beginners, mainly when using small programs. Indeed, modularization makes sense when we use a relatively large program. In our approach, we can teach modularization using each “instrument” in a different module in our program. It naturally links with music, where each instrument is one part of the song.

For instance, in Figure 1, we have a completely transcribed beat that is divided

into six modules. With that, we can avoid the complexity, directing the students' focus on one specific track that contains the activity to be done.

<pre> 1 # Melody 2 G = 67 3 A = 69 4 Bb = 70 5 6 playNote(G) 7 playNote(G) 8 playNote(A) 9 playNote(Bb) 10 11 12 13 </pre>	<pre> 1 # Bass 2 G = 19 3 4 playNote(G, 1.5) 5 playNote(G, 1.5) 6 playNote(G, 2) 7 playNote(G, 1.5) 8 playNote(G, 1.5) 9 10 11 12 13 </pre>	<pre> 1 # Fake 808 2 kick = 1 3 4 playNote(kick, 1.5) 5 playNote(kick, 1.5) 6 playNote(kick, 2) 7 playNote(kick, 1.5) 8 playNote(kick, 1.5) 9 10 11 12 13 </pre>
<pre> 1 # Hi hat 2 for i in range(0, 8): 3 playNote(4, beats=0.5) 4 5 6 7 8 9 10 11 12 13 </pre>	<pre> 1 # Snare 2 snare = 2 3 4 rest(2) 5 playNote(snare) 6 rest(1) 7 8 9 10 11 12 13 </pre>	<pre> 1 # Background synth 2 Ds = 51 3 Gs = 56 4 E = 52 5 6 playNote(Ds, 0.5) 7 playNote(Gs, 0.5) 8 playNote(E, 0.5) 9 playNote(Gs, 0.5) 10 playNote(E, 0.5) 11 playNote(Gs, 0.5) 12 playNote(Ds, 0.5) 13 playNote(Ds, 0.5) </pre>

Fig. 1.: Code Example - Modularization

Parallelism: Usually, parallelism is not taught in an introductory computer programming class due to its complexity, even for those more familiar with computer programming and CS. But when we talk about music and create music with code, it is almost natural to speak of parallelism, where all the tracks can be played simultaneously to achieve one task, producing a complete song.

In Figure 2, we have one example of parallelism, where all the tracks are set to execute (play) simultaneously. So, each track will complete one part of the song, forming, at last, a complete beat.

One additional concept was present in the first version of our curriculum is con-

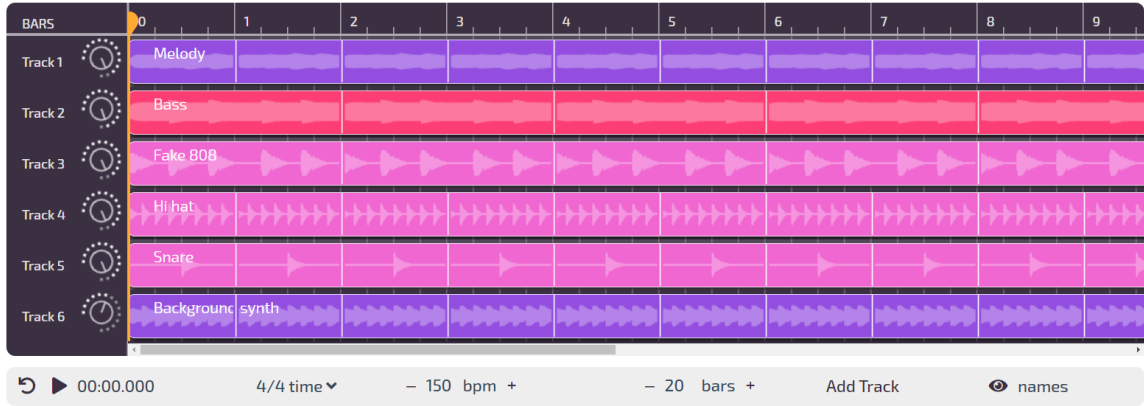


Fig. 2.: Example - Parallelism

ditionals. This concept was taught in the camp reported in Chapter 4 but was later removed from our curriculum. This is indeed a foundational concept in computer programming. Still, as the main objective of *Code Beats* is to engage students in computer programming and not teach all the concepts extensively, we opted to remove concepts that are not naturally connected with music, as is the case of conditionals.

3.2 Genre Choice

One way to instigate interest and motivate one to learn something is to use a topic present in their life. In line with that, one promising approach to engaging all students in CS is using culturally relevant pedagogy [10, 11]. Culturally relevant pedagogy “validates, facilitates, liberates, and empowers ethnically diverse students by cultivating cultural integrity, individual ability, and academic success” [10].

At *Code Beats*, hip-hop was chosen as the musical genre to use to teach computer programming for two main reasons: (1) hip-hop is one of the most popular musical genres, using a vernacular culture, especially for African-Americans and Latino-Americans who are underrepresented in computer science; (2) how a hip-hop beat is created is well suited to teaching computational concepts. Because its creation does not require extensive harmonic knowledge (e.g., Next Episode, from Dr. Dre,

has only one chord), focusing instead on complex rhythms tends to support the more technical concepts present in computational sciences.

3.3 Software Tooling

To put the *Code Beats* approach into practice, it was necessary to select a music-coding platform that was possible to transcribe the songs in a way that they were recognizable. For this, initially, Sonic Pi [27, 38] was chosen, which was used in the experiment reported in Chapter 4. Then, we moved to TunePad [28, 39] used in its first version (TunePad.live)¹, in the experiment reported in Chapter 5 and in Chapter 7, and, in its upgraded version (TunePad.com)², in the experiments reported in Chapter 6 and in Chapter 8.

Sonic Pi was used in the first camp, but it was decided to look for another platform. The most critical point that led to investigating a different platform to implement *Code Beats* is that Sonic Pi must be downloaded and installed as regular software, requiring a basic knowledge of CS. Additionally, until the time the decision to switch was made, it was not possible to install Sonic Pi on Chromebooks. This computing platform is widely used by the *Code Beats* target audience.

The TunePad [28, 39] is a web tool that uses a computational notebook approach, organizing tracks as instruments, each with its code segment, that can be played individually or simultaneously using a timeline. TunePad uses Python³ to create the beats.

As shown in Figure 3, to play notes, students write the commands (1), such as

¹<https://tunepad.live/>

²<https://tunepad.com/>

³<https://www.python.org/>

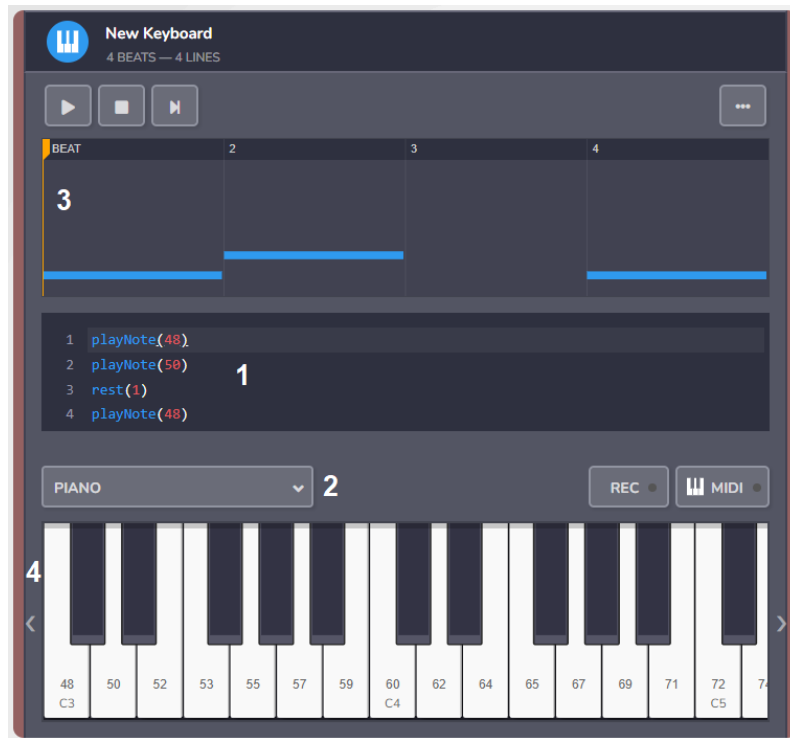


Fig. 3.: Code Example in TunePad.com

command *playNote* using a musical note (initially as a MIDI ⁴ number) as the main parameter. When using a synthesizer, it is possible to choose the instrument (2) that will play in that track. The commands (musical notes and intervals) specified for that segment are visually created (3), showing what will play.

The way that TunePad is built, using a visual style mixed with a textual part, allows the user to hear a musical note or an instrument before using it, by just clicking on it (4). It makes it easier, especially for those not musically trained, to choose a musical note or an instrument that will be played without writing and executing the actual code.

Similarly, as shown in Figure 4, students write the commands (1) informing what

⁴Musical Instrument Digital Interface

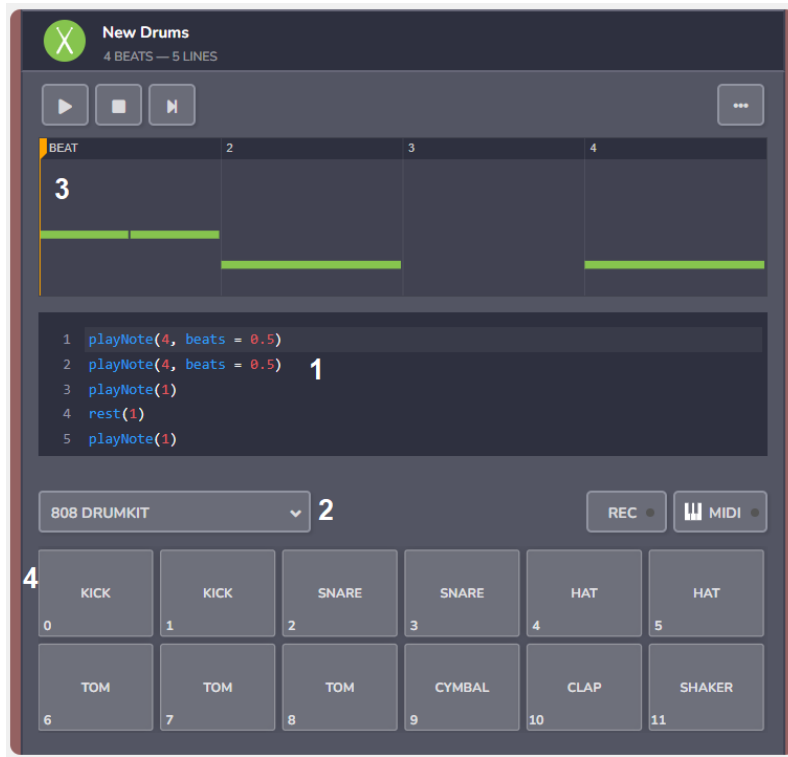


Fig. 4.: Code Example in TunePad.com - Drums

will be played with percussion instruments. In addition, it is possible to select a group of percussion instruments (2), and it is possible to see the track distribution (3) with instruments and intervals. Finally, it is possible to click and hear each instrument individually (4) before using it in the code. Additionally, TunePad is a web tool, making it easier to share projects through the platform without the necessity to save and upload computer files.

3.4 Camp Format

Code Beats was initially planned to be held in person, but due to the restrictions imposed by the pandemic of COVID-19, its format was re-designed to a virtual format. The camps in which the results are reported in Chapter 4, Chapter 5, and Chapter 7 were all virtual.

With the evolution of the pandemic and the ease of restrictions, it was possible to hold two camps in person. The results of the experiments performed in those camps are reported in Chapter 6 and Chapter 8.

The camp whose results are reported in Chapter 4 had 15 classes. All other camps had 10 classes. Their duration and division are reported in their respective chapters. Subsection 3.4.1 reports the contents and their sequence. The first camp was a first exploration, so it was possible to evaluate the content and make some changes, improving the curriculum for the subsequent editions. One of the changes was the duration of the camp. Another difference is in the order in which the contents are taught.

Every class contained some repetitive segments. These segments are reported on Subsection 3.4.2. Additionally, the hands-on activities for the students are described in Section 3.5. At the end of each camp, a beat contest was held, so the students could demonstrate what they had learned.

The pedagogic approach used at *Code Beats* is based on the Use-Modify-Create framework [40]. Throughout the camp, the students receive actual hip-hop songs transcribed to Sonic Pi or TunePad and are instructed to modify one specific track (Modify) to do the activity. However, they are free to explore and use the rest of the project (Use). At the after-class activity and mainly in the camp project, students are instigated to explore their creativity, creating their beats (Create)

3.4.1 Curriculum Distribution

After the first camp, analyzing the main takeaways, the curriculum was improved for the next edition of *Code Beats*:

Camp duration: as four days of the first version were used for revision of the content and preparation for the contest, it was decided to decrease the number of classes to

10, offering the help differently, with after-class help sessions.

Conditionals: after the first experience, it was decided to remove this concept from our curriculum. The main reason is that conditionals are one concept not naturally connected with music. As one of the points of using music as the background is naturally connecting the concepts, it was decided to remove what is not strongly related.

Lists before Repetitions: in our curriculum review, the order in which lists and repetitions are taught was changed, introducing lists before repetitions. The main reason for that is that repetitions controlled by lists are used, so it makes sense to make this change in the sequencing of our curriculum. The final curriculum version is presented in Table 1.

Day	Programming Concept	Music Concept
1	Sequencing	Melody
2	Variables and Constants	System of Musical Notes
3	Functions (with single parameter)	Rhythm
4	Functions (with multiple parameters)	Rhythm and Melody
5	Lists	Chords
6	Repetitions (numeric controlled)	Repetition
7	Repetitions (list iteration)	Repetition
8	Repetitions (nested lists)	Chord progression
9	Modularization	Orchestration
10	Parallelism	Orchestration

Table 1.: Curriculum Distribution - Final Version

3.4.2 Core Segments

Due to its initial format, *Code Beats* was designed to look like a TV show, with some core segments that are “presented” in every class. For the virtual camps pre-recorded videos of each section were used. In the in-person camps, the sections were performed live by their respective instructor.

Coding with Doug: this segment, with its name referring to the author of this dissertation, is where the concepts of computer programming are introduced in depth. The computer programming concept is explained using the programming platform, making the connection with music but without explaining the music concepts. Table 2 lists the videos used in this segment in the camp reported in Chapter 7.

Day	Programming Concept	Link to the Video
1	Sequencing	https://youtu.be/U0zDgYKyZSo
2	Variables and Constants	https://youtu.be/BvIGrhL7yuE
3	Functions (with single parameter)	https://youtu.be/o50LR6Y8k0U
4	Functions (with multiple parameters)	https://youtu.be/0FEZJklpNDQ
5	Lists	https://youtu.be/Ln0Slpj9QoQ
6	Repetitions (numeric controlled)	https://youtu.be/1GFd_dauc14
7	Repetitions (list iteration)	https://youtu.be/RJgm74mRphk
8	Repetitions (nested lists)	https://youtu.be/t6AkQQCLT90
9	Modularization	https://youtu.be/BLrj1fraFrA
10	Parallelism	https://youtu.be/r2wRWPYGHjM

Table 2.: Coding with Doug - Videos

Decoding the Beat: formerly named “Music Theory Minute”, this segment is where the musical concepts are introduced. It is presented by a music professor who collaborates on the *Code Beats* project. Table 3 lists the videos used in this segment.

Day	Music Concept	Link to the Video
1	Melody	https://youtu.be/Gkcr2Z1z0Sc
2	System of Musical Notes	https://youtu.be/bR8y1N01Mso
3	Rhythm	https://youtu.be/QPdmVuduJZ0
4	Rhythm and Melody	https://youtu.be/e8L84M1k9wM
5	Chords	https://youtu.be/XbVts685cWs
6	Repetition	https://youtu.be/zqwbiSN9jiM
7	Repetition	https://youtu.be/N2tPYHLYI44
8	Chord Progression	https://youtu.be/wiBL8PTvp8s
9	Orchestration	https://youtu.be/9Cwb_IoqbAk

Table 3.: Decoding the Beat - Videos

Vocabulary Definitions: at this segment, some of the common vocabulary used in computer programming is presented in a different way, correlating it with everyday situations. Table 4 lists the videos used in this segment.

Along with these core segments, some other sections (i.e., quizzes; complete the lyrics) are present in every class to improve the engagement and students' participation.

Day	Programming Concept	Link to the Video
1	Syntax	https://youtu.be/kdE4qHrS57U
2	Variables and Constants	https://youtu.be/cvob_LkpxT8
3	Functions	https://youtu.be/JSX-KEua77c
4	Parameters	https://youtu.be/hPDpNS7j1kk
5	Lists	https://youtu.be/IVHVzbWhozc
6	Repetitions	https://youtu.be/NAMR-CEp0ms
7	Repetition with Range	https://youtu.be/05fUZWJfk_w
8	Nested Repetitions	https://youtu.be/H-No5rSTGnE
9	Modularization	https://youtu.be/CIpP0VEusF0
10	Parallelism	https://youtu.be/ZvWLS_2JvTY

Table 4.: Vocabulary Definitions - Videos

3.5 Activities

At *Code Beats*, students have an opportunity to put their knowledge into practice with hands-on activities in every class. There are two types of activities. One is the short activity, designed to be solved in 5 minutes, with straightforward solutions strongly connected to the class's content. And the other one is the long activity, which can be solved in 15 minutes or take longer for the most engaged students. The long activity is more open-ended, allowing students to use their creativity more extensively. In the virtual version of *Code Beats*, the short activities were solved during class time and the long activity after class time. Both types were solved during class in the in-person version of *Code Beats*.

The activities are planned to allow the students to code from the very first activity. The way that the songs are coded in Sonic Pi and TunePad allows us to have a realistic-sounding beat that can be changed by the student in a very specific part, without changing the core of its original version, at the same time allowing them to “see” their participation in that beat. Using this approach makes it possible to keep the student’s attention on the concept they are learning in that class, increasing the activities’ complexity in each class.

For example, in the first activity, students are asked to create a melody for a song, guided by the right notes to use. As a result, they can create a melody that matches the actual beat, which is purposely missing in the transcribed one.

Listing 3.8: Code Example 1 - Sonic Pi

```
live_loop :melody do
  #####
  # Create a melody by modifying the code below.
  # To match the background music, use these notes:
  #   Ab4, Bb4, Cb5, Db5, Eb5, Fb5, Gb5
  # To match the dominant chord, emphasize these notes:
  #   Ab4, Cb5, Eb5
  #####
  play :Eb5
  sleep 1
  play :Eb5
  sleep 1
  play :Eb5
  sleep 1
  play :Eb5
  sleep 1
  #####
end
```

For instance, in Listing 3.8, there is one example in Sonic Pi where the students would be asked to read the comments and modify the existing (single-note) melody, which appears at the bottom. By altering the notes, playing the file, and listening to the changes, they begin to understand the implications of their code.

The same with Listing 3.9, an example in TunePad, where the students are asked to complete a melody by choosing the suggested MIDI numbers to match the background beat. The examples above are extracted from an activity. Next, a complete activity is detailed, first using TunePad.live and then adding details specifics to TunePad.com.

Listing 3.9: Code Example 2 - TunePad

```
1  """
2  FIRST TRACK — MELODY
3  Complete the melody using command playNote and MIDI numbers 49 ,
   56 and 57.
4  Your melody should sum up to 4 or 8 beats where each command
   playNote is 1 beat.
5  """
6
7  playNote(49)
8  playNote(56)
```

3.5.1 Example Activity in TunePad.live

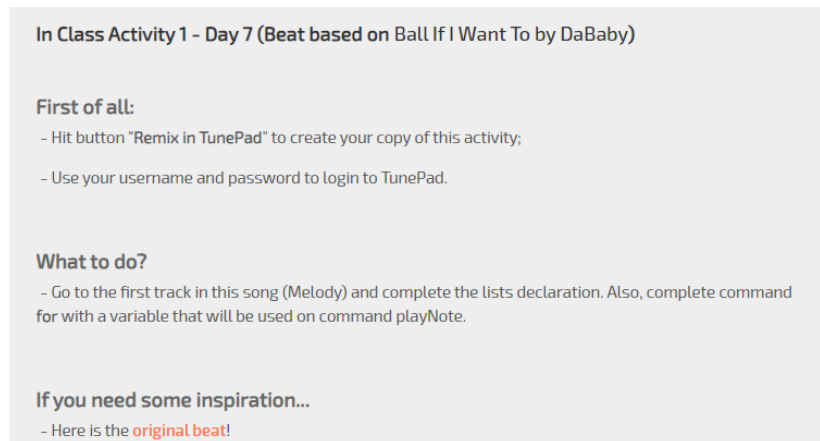


Fig. 5.: In-Class Activity - Example - Instructions

First, in Figure 5, we have the activity instructions, the required steps to create the student copy of the project, and the details about what is expected. This example asks students to complete a list declaration and completes the command *for* with the

variable used in the command *playNote*. The song name, author name, and a link for a song video are also included for students' reference.

BEAT	2	3	4

```
1  """
2  FIRST TRACK - MELODY
3  Complete the lists declaration with constants.
4  Missing notes: D, C, B, B
5
6  Complete command for with a variable
7  that will be used on command playNote
8  """
9
10 D = 74
11 C = 72
12 B = 71
13
14 notes = [ ]
15
16 for ? in notes:
17     playNote(note, beats = 2)
```

**Error: bad token
for ? in notes: on line 16.**

Fig. 6.: In-Class Activity - Example - Melody

After instructions, come the song's tracks. The track where the student must do the activity is the first track. As shown in Figure 6, this example is in the Melody track.

First, the activity instructions are reinforced with more details (lines 2 to 7). Next comes the actual code, where the student has to make the changes. In this example, the student must complete the list declaration (line 14) with the already declared constants in the order stated in the instructions (D, C, B, and B). Additionally, the student must replace the question mark (line 16) with the variable that is being used on the command *playNote* (line 17). With this activity, we expect to see if the student understands how a list works and how a repetition with lists works, changing the variable's value at each iteration.

Next comes the additional tracks that form the complete song. The student does not have to worry about these tracks but is free to explore and make changes. Each track corresponds to one instrument. Looking at it individually, it appears to be simple, and listening to a single track may not make sense, but it forms a complete and recognizable song when all tracks are played together.



Fig. 7.: In-Class Activity - Example - Bass

In Figure 7, a bassline is shown where three different notes (lines 1 to 3) are used. First comes a pattern that repeats three times (lines 6 to 11), lasting for two measures (4 beats) that first plays the musical note B (line 7), rests for five beats (lines 8 to 10), and then plays the musical note C (line 11). Note that in this example, two commands *rest* are used in sequence. That is to group the commands in a measure (4 beats) in line with the musical concept. Next comes a measure that contains the musical note B played for two beats (line 14) and an interval of two beats (line 15). The last measure for this track is formed by musical note D played for three beats

(lines 17 and 18) and musical note C played for one beat (line 20).

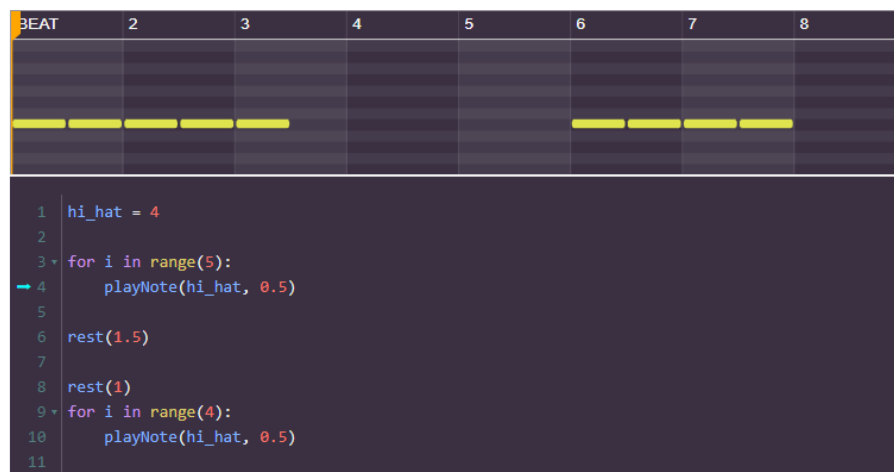


Fig. 8.: In-Class Activity - Example - Drums (Hi-hat)

In Figure 8, a simple hi-hat track first plays the hi-hat five times during 0.5 beats (lines 3 and 4). Then rests for 2.5 beats (lines 6 to 8) and finishes playing the hi-hat four times during 0.5 beats (lines 9 and 10).

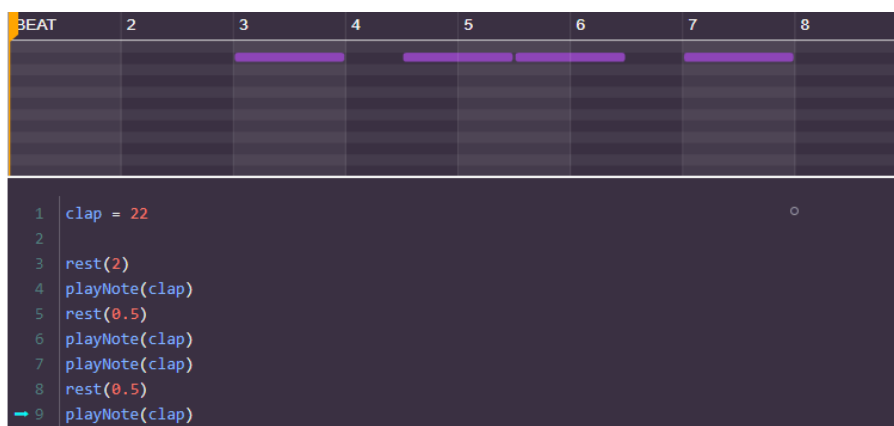


Fig. 9.: In-Class Activity - Example - Drums (Snare)

Next, in Figure 9, we have an additional drum track that plays a clap with some intervals, creating a rhythm. The command *playNote* plays the constant clap for one beat in lines 4, 6, 7, and 9. The command *rest* adds an interval of two beats on line 3 and an interval of 0.5 beats on lines 5 and 8.

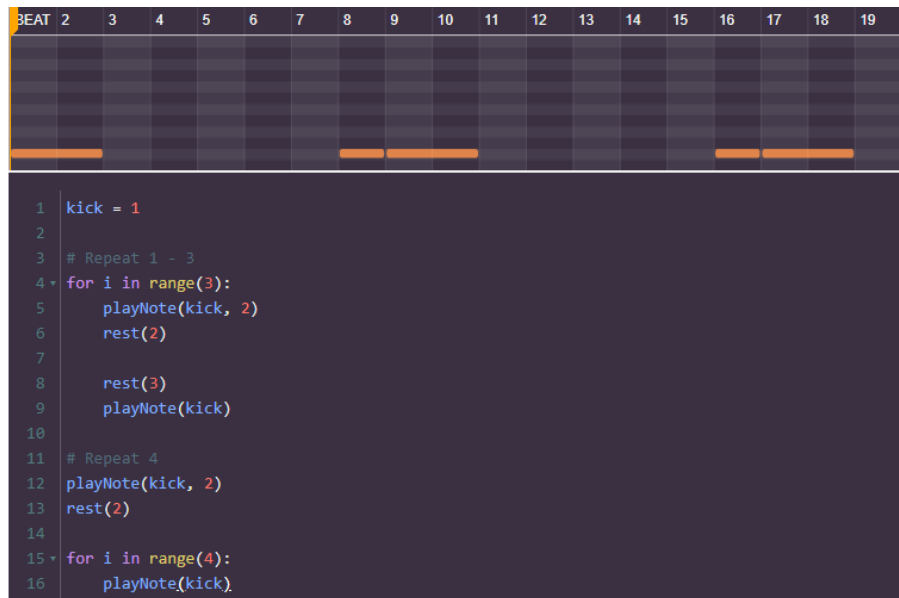


Fig. 10.: In-Class Activity - Example - Drums (Kick)

Finally, Figure 10 shows the last drum track. It plays the drum kick in a pattern three times (lines 4 to 9), adding up to six measures, playing the drum kick for two beats (line 5), resting for five beats (lines 6 to 8), and then playing the drum kick for one beat (line 9). Next, play the drum kick for two beats (line 12) and rests for two beats (line 13). Finally, play the drum kick for one beat four times (lines 15 and 16).

In addition to playing each track individually, students can play all the tracks simultaneously, reproducing the song similar to the actual beat on which the activity was based. In Figure 11, there is the timeline for this example where the “Melody” track starts playing in the first beat, and all the other tracks start playing in the 9th beat (third measure).

In this example ⁵, students can explore all the concepts that they are learning during the camp, functions, parameters, and constants in all tracks, variables in tracks that have repetitions, lists, and repetitions with lists in the “Melody” track,

⁵Link for the activity: <https://tunepad.live/app/dropbook/53401>

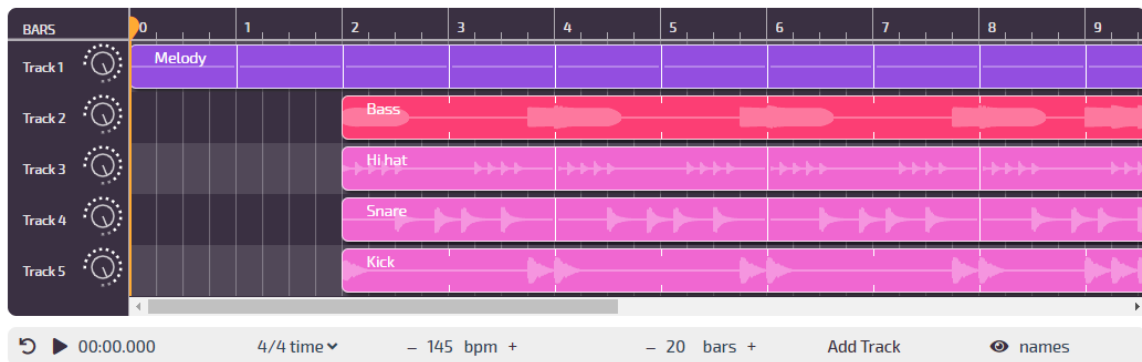


Fig. 11.: In-Class Activity - Example - Timeline

numerically controlled repetition at the “Bass”, “Hi-Hat” and “Kick” tracks, modularization as we are keeping the code in small parts and parallelism, playing all the tracks at the same time.

3.5.2 Example Activity in TunePad.com

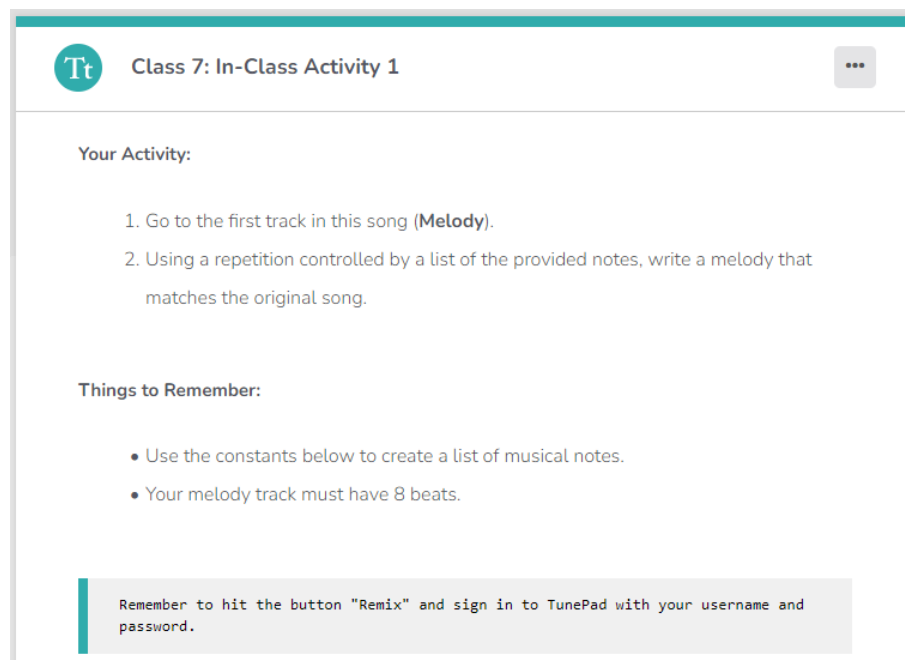


Fig. 12.: In-Class Activity - Example 2 - Instructions

Beyond the clear difference in the visual interface between TunePad.live and

TunePad.com and the engine behind the platform, which will not be discussed here, we took advantage of the platform migration to improve *Code Beats* activities in terms of instructions and objectives. With that in mind, only the parts that have differences will be explained in this subsection and not the whole project ⁶.

To start, in Figure 12, we modified the instructions to have clearer instructions and tips for doing the activities. Additionally, we reinforced that remixing the project to create the student's own copy is necessary.

With the platform upgrade, it is possible to embed videos. So, instead of having only the link for the video that contains the original song, the video can be played from the project. Additionally, as it is possible to observe in Figure 13 we added a video with the expected result from the track that the student will work.

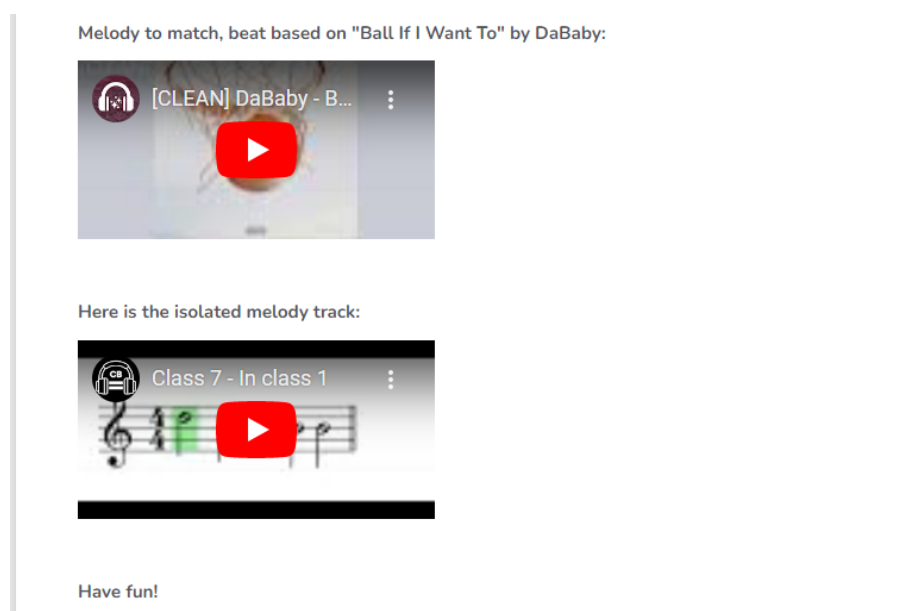


Fig. 13.: In-Class Activity - Example 2 - Videos

After experiments reported in Chapter 7 we decided to change the activities' framework, not providing partial code to be changed or repaired. Rather, the activi-

⁶<https://tunepad.com/project/38124>

ties ask students to create their own code. In some cases, the activities are still with the constants already declared. Figure 14 shows an example of an activity track the students have to solve. First, we have the activity instructions (lines 2 to 9). Lines 8 and 9 exemplify the syntax to create repetition. Next are the constants declarations (lines 12 to 14) with their corresponding MIDI numbers. Then the space to students builds their own code.

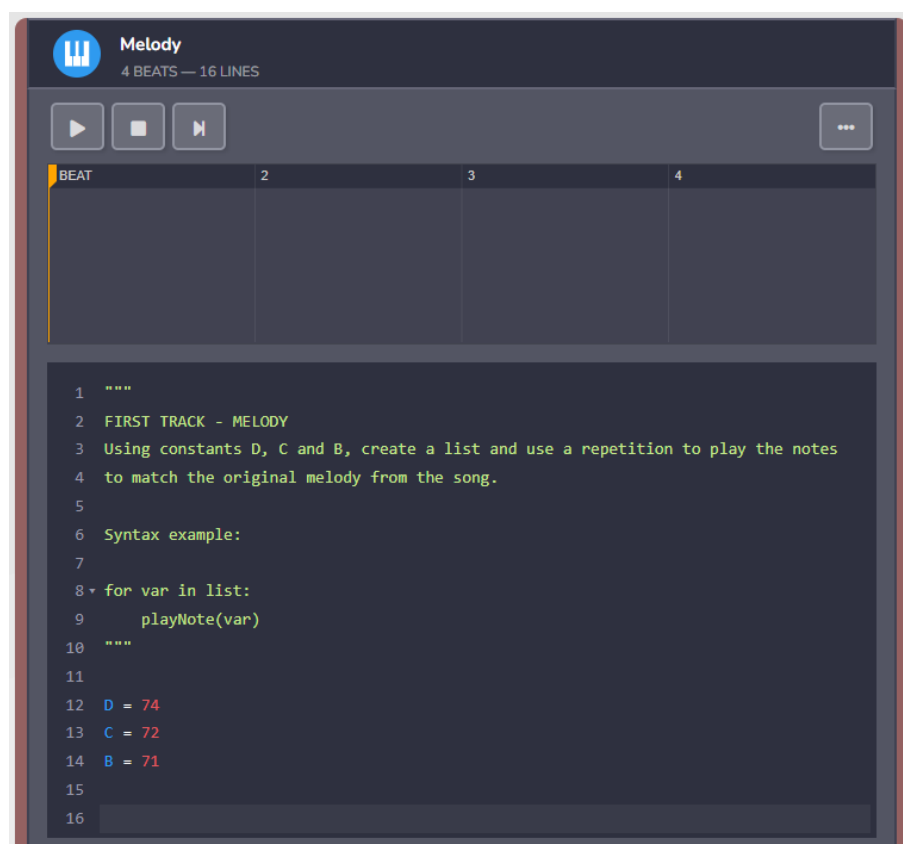


Fig. 14.: In-Class Activity - Example 2 - Melody

3.5.3 List of Activities

Beyond the hands-on activities for the students, in each class, before students start doing the activities, it is explained how to solve a similar problem using a worked example. This worked example is similar to the short activity and gives

students guidance about how to solve the activities. All the worked examples, in-class, and after-class activities using TunePad.live can be seen in Appendix A. All the worked examples, short activities, and long activities using TunePad.com can be seen in Appendix B.

CHAPTER 4

MEASURING ENGAGEMENT TOWARD COMPUTER SCIENCE

Despite all the efforts to broaden the participation of underrepresented groups in CS, a difference in the proportion of the overall population and the population studying or employed in the field still exists, where groups such as African-Americans and Hispanics or Latinos are underrepresented in the area [3, 4]. As presented in Chapter 2, this difference persists even in schools that formally offer opportunities for students to take CS classes [16]. One reason is that the approaches that try to attract and engage students use incentives that work with those who are already interested in the field, such as robotics and video games, perpetuating the stereotype of CS as a singularly focused activity [1].

One non-conventional way to teach code that seems to engage those not interested in CS is using music to teach computer programming [12]. Furthermore, using the musical genre that is part of the culture of populations underrepresented in CS, such as African-Americans and Latinos, can enhance its potential [10].

I applied *Code Beats* in a pilot camp, with students that are in middle school, to measure the effect on student engagement toward CS. In this pilot camp, described in this chapter, I experimented with the first version of *Code Beats* curriculum, in a three-week-long virtual camp using Sonic Pi. This experiment’s main objective was to analyze our approach’s impact in terms of engagement toward CS.

This pilot edition of *Code Beats* program was organized as a three-week online camp (note: due to COVID-19), with a one-hour class held each weekday. Classes

were streamed to students using the Twitch¹ platform, where the students could interact with each other using the platform’s chat. Students gave feedback to instructors using the multiple choice, and open-ended questions presented during lectures using the online presentation platform Mentimeter², in addition to end-of-class interactive quizzes and after-class help sessions. Each day students had an after-class programming assignment to reinforce the knowledge about the content learned so far. The students were strongly encouraged to upload their submissions each day, as assignments were reviewed daily to track students’ progress.

4.1 Research Questions

The main purpose of this study was to determine whether using culturally relevant music to teach computer programming to students in middle school would affect their engagement and attitude toward Computer Science. In order to validate the approach, we investigated two research questions:

RQ1: How does the use of hip hop in teaching coding affect the engagement (i.e., enjoyment, confidence, belonging, intent to persist) of middle school students towards CS?

RQ2: Does this graded scaffolding approach enables students to progress to the **Create** phase of the **Use-Modify-Create** framework?

¹<https://www.twitch.tv/>

²<https://www.mentimeter.com/>

4.2 Methods

4.2.1 Participant Demographics

Because the camp was conducted online, recruiting was conducted via social media, with students registering by purchasing a one-month channel subscription (\$25) on Twitch. Over the course of the three weeks, an average of 45 students attended the *Code Beats* camp. From this group, 31 students answered the pre and post-survey and had IRB parental consent. From this group, 17 students had no previous exposure to *Code Beats*, and so these 17 students are the only students included in the questionnaire results. This population was, on average, 12 years old (min: 10 - max: 14), and was 70.6% male and 29.4% female. 41.2% were self-declared minorities, and 23.5% were Black. 64.7% of the students declared that they knew how to read music, and 41.2% declared that they had previously taken a programming class.

4.2.2 Data Collection

The primary way we measured students' perception of Computer Science was by asking them to answer a survey at the beginning of the first day of class and at the beginning of the last day of class. The survey included 19 questions answered via a Likert scale, from 1 (Strongly Disagree) to 5 (Strongly Agree). During the class, students also provided live feedback via Mentimeter, a crowd interaction tool. To provide qualitative support for our quantitative data, we performed an open coding process on these written responses, grouping them by theme.

At the end of each day of class, the students were asked to do an activity, designed to take 15 minutes to complete. They were encouraged (not required) to submit their daily activity from which a subset was used to provide constructive feedback during

the next day’s class. On the last day of camp, we conducted a beat contest, where each student could submit an original hip-hop beat created with what they had learned during the camp. They were informed about the contest on the first day of camp.

4.2.3 Data Analysis

To compute the statistical significance of the pre and post-camp survey results, the Wilcoxon signed-rank test was applied to each question individually. The Wilcoxon signed-rank test is a non-parametric statistical procedure for comparing two samples that are paired or related. The Wilcoxon signed rank test does not require that the data is normalized [41]. With Wilcoxon signed rank test, if the p-value is lower than 0.05 ($p\text{-value} < 0.05$), we can say that we have statistically significant evidence to reject the null hypothesis (no changes in answers between the first and last survey).

Additionally, the Wilcoxon signed-rank test was applied to groups of categorized questions. For that, the questions in the survey were placed in four groups: computing enjoyment (5 questions); computing confidence (5 questions); identity and belonging in computing (5 questions); and intent to persist in computing (4 questions). Note that, because some answers were positively phrased and others negatively phrased, the value of the negative statements was inverted prior to analysis. For example, a 5 (Strongly Agree) answer to the question “Computers are not for me” was converted to a 1 (Strongly Disagree) to be compatible with answers to the question “Computers are cool”. Then the answers to the questions of each group for each respondent were added, and the total for pre and the total for the post was considered to calculate the p-value.

4.3 Results

4.3.1 RQ1: Student Engagement

When questions were grouped, as described in Section 4.2, all four categories of questions showed a statistically significant difference ($p\text{-value} < 0.05$) between pre and post-surveys, as shown in Table 5.

Category	p-value
Computing enjoyment	0.006
Computing confidence	0.001
Identity and belonging in computing	0.011
Intent to persist in computing	0.022

Table 5.: Differences between pre and post-surveys

Drilling down to individual questions, we show positively-phrased questions in Figure 15. As you can see, when moving from the pre-survey (top) to the post-survey (bottom), students' answers became more positive, as indicated by the increasing amount of *Agree* and *Strongly Agree* statements, shown in green. Eight of these twelve questions showed a statistically significant difference when analyzed individually (p -values to the right of each question).

In Figure 16, we show only negatively-phrased questions, such as “I cannot be a computer scientist”. When moving from the pre-survey (top) to the post-survey (bottom), students' answers changed, becoming more negative, as indicated by the increasing amount of *Disagree* and *Strongly Disagree*, shown in red. Three of these seven questions showed a statistically significant difference when analyzed individually.

Further supporting this point, other data shows that students remained engaged throughout the three-week camp. During the 3 weeks, students were assigned 14

activities, where 7 were open-ended activities, including the beat for the contest. On average, 10 students submitted an assignment even though it was not required, with students submitting a total of 140 assignments over the course.

When asked on the final day “Do you think you might take a programming course in school?” students had overwhelmingly positive answers such as “*yes because I love it #CodeBeats forever*”, “*Definitely!! I really enjoy coding and computer science, and I intend to pursue it*”, and “*yes because I want to have a higher chance to get into the tech center*”. Even students that were less confident in their future were positive about their experience, saying “*maybe because I definitely have fun doing this*”, “*I might, because it is really fun*”, and “*I think I could*”. Taking into account this qualitative data along with the above quantitative data, we can answer RQ1, noting that using hip hop to teach computing concepts does lead to an increase in engagement.

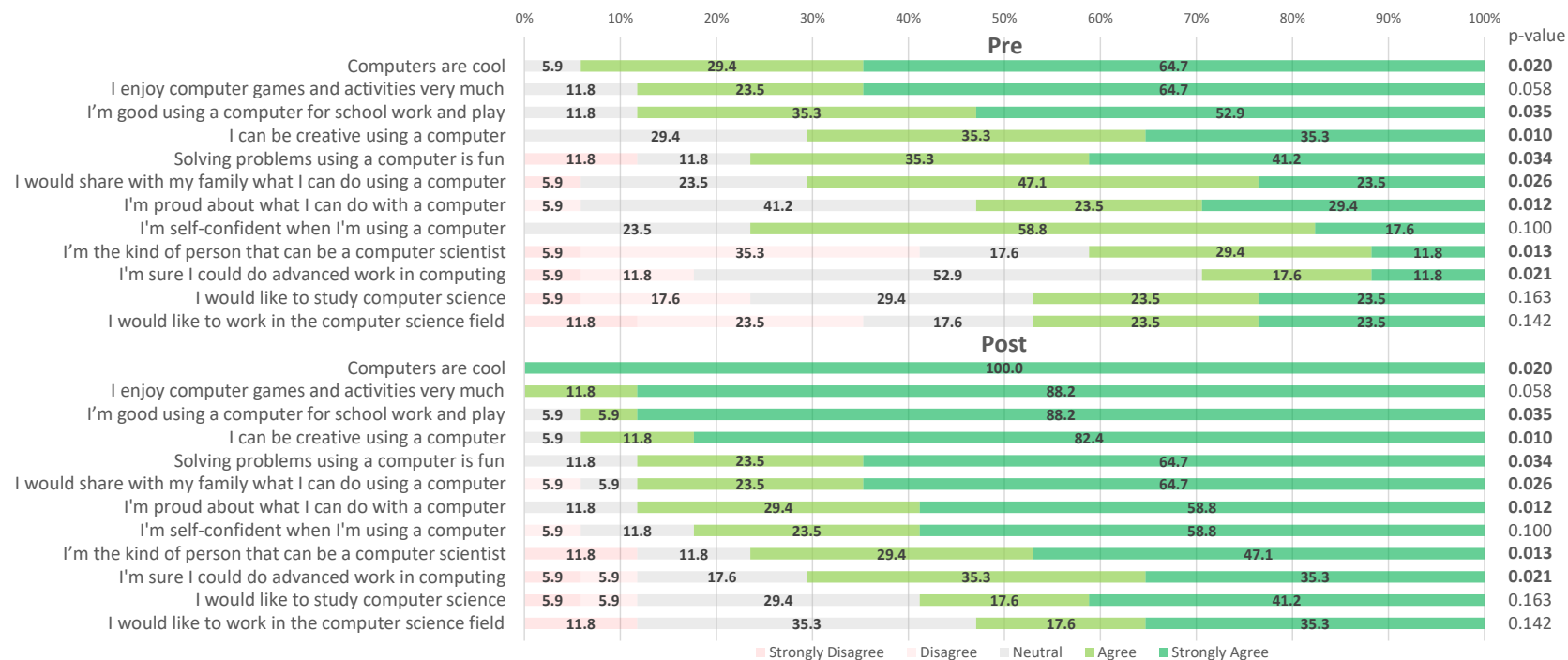


Fig. 15.: Survey Results: statements with a positive view of computer science

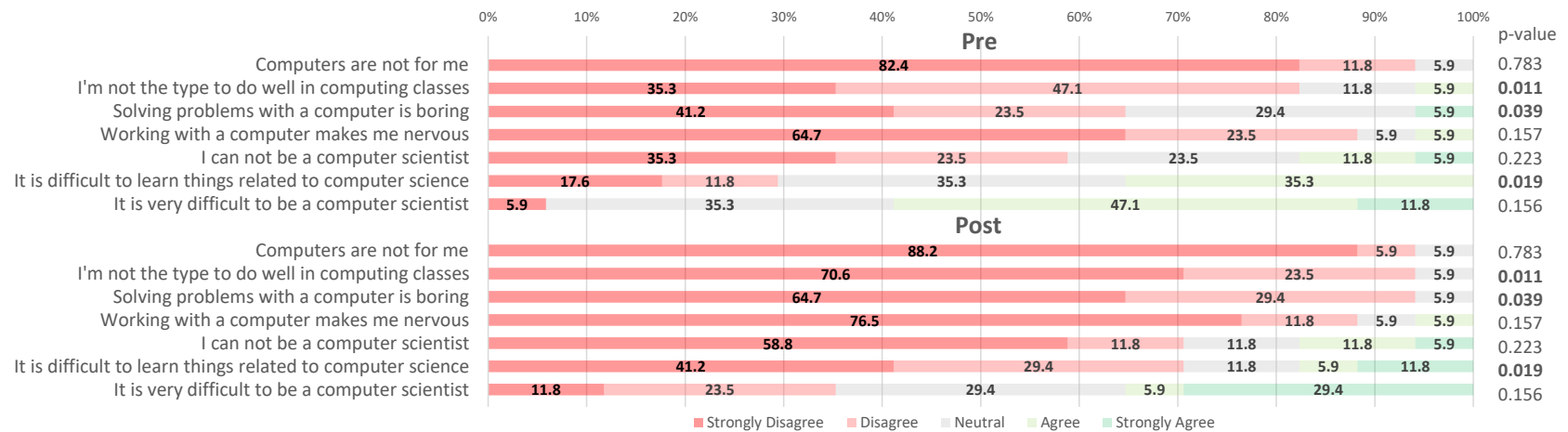


Fig. 16.: Survey Results: statements with a negative view of computer science

One potential caveat is that students may become more engaged with CS after taking any CS class. While this may be true, we believe that many of the students in our class would *not* have taken other CS classes. When asked, on the first day, why they signed up for our course, many responses directly referred to hip-hop and music, saying “*I wanted to learn code, and I love rap*”, “*I’m very interested in beats like Dr. Dre, and I saw this, so I want to make or create beats*”, “*I wanted to learn how to make music with code*”, and “*For fun and also learning to code with music. Also, I love music, so I was like, why not make it.*”.

4.3.2 RQ2: Developing Creators

If students can create an original beat by the end of the camp, they may have achieved the highest level of learning in the Use-Modify-Create framework. To investigate this RQ I analyzed the beats students submitted for open-ended assignments (i.e., assignments without pre-defined beats) over the course of the camp. From these submissions, I only considered original beats; filtering any beats that were even partial copies from earlier assignments.

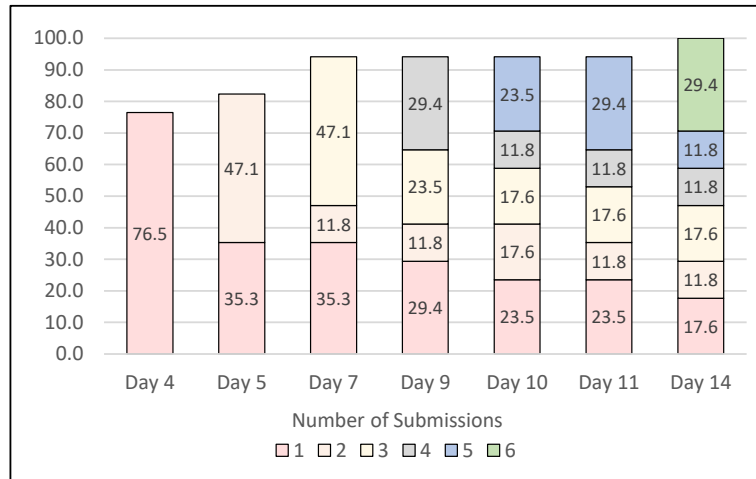


Fig. 17.: Number of original beats per student over time

As it is possible to see from Figure 17, we have graphed the percentage of students that contributed an original beat over time. By Day 4 (the first submission of an open-ended activity), 76.5% of students contributed an original beat, and by Day 14, 100% submitted at least one beat. I prepared a playlist³ of the beats submitted on Day 14, as we believe that their overall quality indicates students’ relatively deep learning. Note that many students submitted more than one original beat, which we also show in Figure 17, with 29.4% of students submitting six original beats by Day 14.

4.4 Limitation and Threats to Validity

Although the findings of this experiment are promising and highlight its positive effect on students’ engagement with computer science, there are several validity concerns to address. Specifically, the sample size of students included in the study was smaller than the total number of students who participated in the camp. This was due to the exclusion of students who attended a pre-pilot camp.

It’s important to take into account external factors as well. Since our classes were virtual and streamed, we couldn’t oversee the students’ learning environment or any potential factors that could affect it.

4.5 Conclusions

Based on the survey’s results and beats submitted during *Code Beats* and for the contest, I can say that the use of hip-hop to engage students in computational thinking is promising. It is possible to note a statistically significant change in the engagement towards computer science, mainly when comparing the questions in a grouped way. Also, it is possible to mention that the use of the Use-Modify-Create

³<https://bit.ly/CodeBeatsPlaylistContest>

framework was successful, as all 17 students submitted at least one beat of their own creation.

Additionally, there are some points that can be highlighted: (a) The importance of the beats contest: A motivation from students to create and show their beats at the contest was noted. It was noted during the classes via chat and with numbers with beats submitted; (b) The quality of beats: at the end of the third week, the students were able to create their own beats, as we had some very good beats, analyzing via musical perspective, and we had some very good beats analyzing via the code perspective and the use of the Sonic Pi resources.

Despite the good results from this pilot camp, there are some limitations in our work: (a) The demographic of our students was not ideal, as we were planning to have more Black and Latino students as our target audience; (b) Sonic Pi has to be installed or used as standalone software; it might need some previous knowledge from students about how to do that. Also, there is some difference between platforms (i.e., Windows and Mac); (c) Due to the COVID-19 outbreak, this pilot camp was held online. It was initially planned to be held in person, and it was initially a limitation, but it also opened doors and new ways that were not being considered before.

CHAPTER 5

INDIRECTLY ENGAGING ADULT LEARNERS

Motivated by the results presented in Chapter 4 and the overall experience with *Code Beats*, I decided to experiment with *Code Beats* with a different audience, adult learners.

Teaching computer programming to adults can empower this immense and fast-growing population, improving their quality of life and financial outlook [42]. In fact, for the subset of adults who truly embrace computer programming, learning these skills can be life-changing. The field has, for years, had a considerable number of open positions that will increase for at least the next ten years. The U.S. Bureau of Labor Statistics [2] projects that the number of CS-related posts will increase by more than 667,000 positions, or 13.4%, from 2020 to 2030.

Unfortunately, it is challenging to attract adults to computer programming. They typically view computer programming as difficult and boring, something impossible for them to learn at this stage of life [1]. Getting these skeptical adults to even sign up for a computer programming class requires some convincing. However, if this initial reluctance can be overcome, there is hope, as even a brief, concrete learning experience with computer programming has been shown to significantly impact adults and their attitudes towards programming [1].

To attract people resistant to a particular field, they may need to be indirectly introduced to it. Referring to the field of mathematics specifically, for instance, Seymour Papert said “*The mathophobia endemic in contemporary culture blocks many people from learning anything they recognize as ‘math,’ although they may have no*

trouble with mathematical knowledge they do not perceive as such.” [43] As presented before, one indirect approach to teaching computer programming, which may help people overcome their hesitation, is through music [26, 29, 33]. If students are programming music, it turns out that much of their focus is on musical creation [28], and they learn the necessary computational concepts almost by accident. When culturally relevant music, such as hip-hop, is used, it creates an even more powerful platform for students to learn computational concepts by engaging in the seemingly unrelated but extremely engaging task of creating beats [10, 11].

This chapter reports the results of an informal computer programming course that presented foundational computer programming concepts to an audience formed by adult learners, coding hip-hop beats. This course was held virtually for five weeks, with one hour class twice a week, using TunePad.live and the second version of *Code Beats* curriculum (refer to Chapter 3, Table 1). During this course, I investigated the impact of this indirect approach, already known to be effective with young learners, on the engagement and motivation of adult learners.

5.1 Related Work

To date, most research that creates and investigates ways to motivate and engage students in learning foundational concepts of computer programming targets young learners. This body of work typically analyses the effects of tools and methods of programming among students from primary and secondary educational or, at most, introductory college-level classes. While an increasing number of for-profit coding boot camps target adults, these courses are expensive and require full-time engagement for about twelve weeks [44]. Unfortunately, most adults are either unwilling or unable to dedicate much of their resources to what they perceive as a high-risk endeavor [45]. Furthermore, some boot camp students faced the same informal com-

munity boundaries (e.g., race, gender, previous experience, and stereotypes) as in other informal and formal learning environments [46].

Some studies have begun to investigate interventions with adults. For instance, Charters et al. [1] report an experiment that uses an educational video game to provide adult participants with programming experience. In this work, the authors found that, after a positive exposure to programming, through the educational video game, attitudes such as the belief that programming is difficult, boring, and something that they could not learn, changed positively [1].

Using an online survey, Guo [42] investigated older adults' motivation to learn computer programming and the frustrations they experienced. In terms of motivation, three important categories were found: (a) Age-related Motivations: respondents wanted to make up for missed opportunities during their youth; (b) Enrichment-Related Motivations: for instance, learning programming to implement a specific hobby project idea; and (c) Job-Related Motivations: for instance, respondents wanted to learn programming as continuing education that is relevant to their current job or to improve their future job prospects. Finally, in terms of frustrations, findings were categorized into (a) Age-Related Frustrations: for instance, cognitive limitations such as bad memory; (b) Pedagogy-related Frustrations: for instance, lack of instructional scaffolding; and (c) Technology-related Frustrations: for instance, debugging syntax and run-time errors.

Krafft, Fraser, and Walkinshaw [47] investigated the effects of using Scratch, a block-based language already known as an approach that motivates young learners with adult learners. They observed a positive effect on students' self-perception and motivation to continue learning programming.

In a longer study, during a six-month-long course with textual and visual programming languages, Sayago and Bergantiños [48] examined the computer program-

ming learning experience of a group of older adults and active computer users with low levels of formal education and no previous experience with computer programming. By the course’s end, participants learned and understood how to write simple programs.

There are several methods and tools to engage and attract people to computer programming, which vary from unplugged activities [17] to the use of robots [18] to the use of block-based environments [19, 20, 21] to the use of video-games development [22] and the use of music creation [26, 12, 27, 28], but as said before, the vast majority targeting the young part of the population.

5.2 Research Question

This study aims to analyze this learning experience’s influence on adult learners’ perceptions of computer programming. In line with that, the research question being investigating is: How does the use of hip-hop in teaching coding impact adult learners’ perceptions of computer programming?

5.3 Methods

5.3.1 Study Design and Data

This chapter reports on data collected during a 5-week virtual course. During that period, participants attended streamed 1-hour lessons offered twice a week. The classes were live-streamed using the YouTube ¹ platform, where the participants could interact with the instructors and each other using the platform’s chat. Each class consisted of a mix of interactive live sections and pre-recorded coding and music lessons, with two hands-on activities completed during class and one activity completed after

¹<https://www.youtube.com/>

class. Every activity was designed to explore the concept taught that day but could also include previously introduced concepts, adding more complexity to the activities each day. The background to these activities was a realistic-sounding beat based on an actual hip-hop song. College musicians transcribed that into TunePad. At the end of the course, participants engaged in a beats exhibition and contest, where they created their beats to show what they learned. After the course, they were asked through an online survey to answer a set of open-ended questions:

Q1. What did you think about computer programming before *Code Beats*?

Q2. What do you think about computer programming now?

Q3. After *Code Beats*, are you interested in learning more about computer programming? Why or why not?

Q4. After *Code Beats*, are you interested in a career in computer science? Why or why not?

Q5. Did the use of music in *Code Beats* change what you thought about computer programming? If so, how?

5.3.2 Participant Demographics

To reach participants, this session of *Code Beats* was organized and publicized by Computer CORE ². This organization prepares under-served adults in Virginia to realize career aspirations with foundational digital and professional skills. During the five-week course, an average of 40 adult learners attended each class. From this group, 32 participants answered the post-survey, which was not mandatory. This population was from a wide range of ages, where the youngest participant was 20 years old, and the oldest participant was 74 years old (average age was 43.8). Of those,

²<https://www.computercore.org/>

37.5% identified as man, and 62.5% identified as woman. In addition, 59.4% were self-declared minorities, and 43.8% were African-American. In addition, 34.4% of the participants knew how to read music, and 18.8% had previously taken a programming class. Hip-hop is the music genre preferred by 31.25% of the participants.

5.3.3 Data Analysis

To answer the research question, the responses from the five questions stated above were analyzed using a Reflexive Thematic Analysis, also known as Braun & Clarke [49], in an inductive and semantic way, where the coding and theme development reflects and were directed by the content of the data.

First, all the responses were grouped by question in a spreadsheet and were de-identified, where a pseudo-identifier was attributed to each learner. Then, this spreadsheet was shared with the author of this dissertation and a research assistant, so each could start creating codes independently. In the first cycle, an open coding process was used, where every answer was analyzed, and one or more codes were attributed when applicable. After that, a new code was created whenever necessary, as determined by the answer's content.

Next, the two independent researchers met to merge and agree upon common codes in the second cycle. At this point, the following actions were taken: (a) For exactly the same codes, the code was maintained; (b) For codes with different words but with the same meaning, the codes were merged; (c) For codes identified by only one of the researchers, the question and the answer were re-analyzed by both, and after discussion, they decided whether the code should be deleted or kept. After finishing this process, the remaining codes were analyzed to identify themes emerging from the codes.

5.4 Results

In this section, the results of the reflexive thematic analysis are presented separated by question from our survey (Q1 to Q5). For each question, the themes from the open coding process and how many times that theme appeared in the data are listed.

Q1. What did you think about computer programming before *Code Beats*?

The responses to this question were grouped in two themes, *Theme 1* (left): *Coding would be difficult/impossible/boring to learn* and *Theme 2* (right): *I was curious about computer programming*. Figure 18 presents all of the codes that contributed to these themes and the number of times each code occurred.

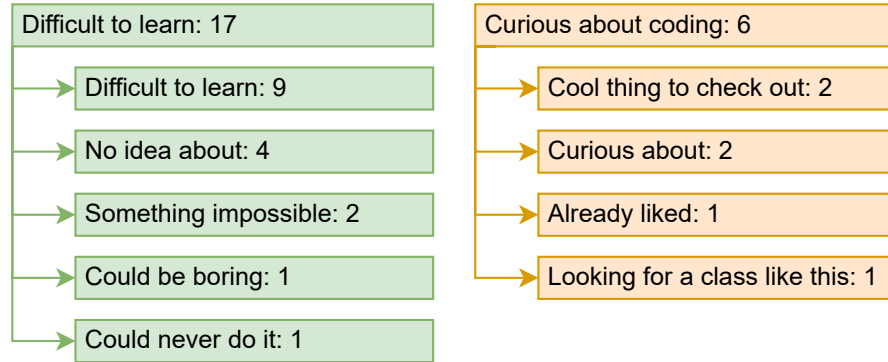


Fig. 18.: Q1 - What did you think before?

Theme 1: Difficult to learn (n=17) - Participants were initially intimidated by the prospect of learning programming, believing that it was, at best, difficult and sometimes impossible for them to learn. Most adult learners had reservations about their ability to learn programming before taking *Code Beats*, as shown by this group of codes that appeared 17 times in the responses for Q1. Participants' responses included: "I thought [coding] was something **impossible** and that I could never do it" (S12). "I thought [coding] was **difficult and boring**" (S21). "[Coding] was **com-**

plicated and beyond my ability” (S24). “I thought [coding] was fairly **difficult to learn**. It also seemed like it would involve a lot terminology I wouldn’t understand and prerequisite skills.” (S29).

Theme 2: Curious about coding (n=6) - While most adults were intimidated by computer programming, a few adult learners were curious about computer science. For many of these curious adult learners, the fact that *Code Beats* was centered around music seemed to make it less intimidating, or at least potentially fun, to them, with relevant responses appearing a total of 6 times for Q1. Participants said: “Had no idea how to do it. Just sounded like a **cool thing to check out**” (S2). “I did not understand how to code, and make music out of Python. I am a music producer, and would **like to better understand** computers, so I figured that this is the best fit for me” (S28). “..then I saw my child to simple coding games and toys and programs and *Code Beats* seemed like a fun way to **try it out**. I really enjoyed it and my big break through moment was when I realized that the errors were my friends and showed me exactly what to change. Before that I saw it as me getting it wrong. That little shift in mindset made it so much more fun to **try out things**” (S12).

Q2. What do you think about computer programming now?

The responses to this question were grouped into four themes. The codes are shown in Figure 19, from left to right, and from top to bottom: “I’m confident that I can learn computer programming” (10); “I’ve improved my knowledge about computer programming” (9); “I’m interested in learn more and use it in the future” (9) and “Programming classes are not for me / it is difficult to learn” (5).

Theme 3: Increased confidence (n=10) - It seems that taking the class increased the learners’ confidence and helped them to understand that, while it might take practice, computer programming was a skill they could learn. This theme

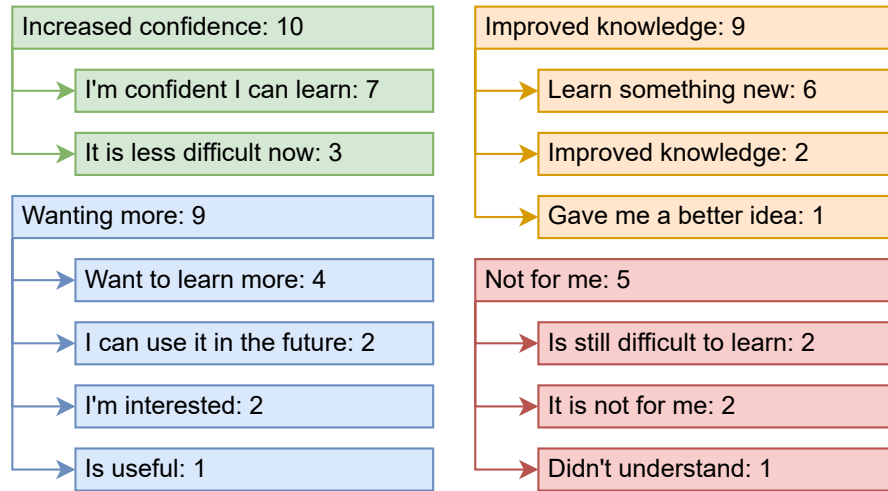


Fig. 19.: Q2 - What do you think now?

emerged from a group of codes that appeared 10 times in the responses for Q2. Some examples of responses that we received for this question and were grouped in this theme are: “I think if I put the time to learn into it, **I can do it**” (S7). “**I think it is possible to get good at it** with lots of practice. I think I’ve found my inner geek!” (S12). “I think that **I am capable of doing computer programming**. It doesn’t seem as hard as I would think and *Code Beats* taught me that!” (S19). “It’s still a bit complicated but seems **less daunting**. I think **the basics are doable** with continual practice” (S29).

Theme 4: Improved knowledge (n=9) - This theme emerged from a group of codes that appeared 9 times in the responses for Q2. Participants said: “I see **how to use it better** and how it applies to so many things. I loved using TunePad to make a song” (S2). “Honestly I don’t see Myself as an Older Job Seeker being a Computer Programmer. The Course Has **Added a More In depth layer to my core skill set**. The Easier YouTube Platform makes For Much Easier Access” (S16). “I can understand and **know the vocabulary of coding** which helps in my next computer coding class” (S18). They used many growth phrases (e.g., “use it better”,

“added to my skill set”, etc.) to describe how they felt the class had changed their knowledge of coding.

Theme 5: Wanting more (n=9) - Adult learners who attended this class may have increased their interest in pursuing further education and even employment in the field. This theme emerged from a group of codes that appeared 9 times in the responses for Q2. Some examples of responses that we received for this question and were grouped in this theme are: “I might be **interested in it now**. In fact I might also **think about pursuing a career in it**” (S1). S20: “**Excited to do more** and do my own job as a *Code Beats* master”. (S20). “**I would like to get certified** in Python and A+. I would like to also get Security+ ,and Network+” (S28).

Related to this theme, while we could not track all the participants after the class, we know 18 participants who followed up this class by attending a Python class and at least one that gained employment in the field.

Theme 6: Not for me (n=5) - As expected, a few adult learners decided that computer programming was not for them. This theme emerged from a group of codes that appeared 5 times in the responses for Q2. Some examples of responses that we received for this question and were grouped in this theme are: “**Confused**. Programming as I saw in *Code Beats* is very different from, say, Data Structures in C. But it seems to be fun.” (S27). “**Not in the computer programming class.**” (S32).

Q3. After *Code Beats*, are you interested in learning more about computer programming? Why or why not?

The responses to this question were grouped into two themes: “I would like to learn more about computer programming. I can do this.” and “I’m not interested in computer programming.”. The codes and their counting is presented in Figure 20.

Theme 7: Ready for more (n=25) - Many adult learners were interested

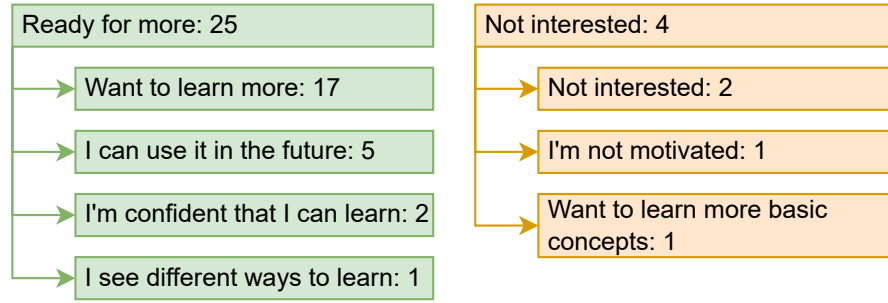


Fig. 20.: Q3 - Interested in learning more?

in programming and optimistic about their ability to learn it, with responses from this theme appearing 25 times for Q3. Participants said: “Yes. I think **I could enjoy having a job doing computer programming**” (S7). “**Yes very eager**” (S15). “Yes, **I was very much interested in learning more**. I thought the *Code Beats* class was fun and it was an introduction to Coding. So, it makes me **feel confident** that I could take on a Computer Programming course” (S19). “Yes **I’m more interested in coding** because I see there is a different way to learn the skills needed” (S21). “**I feel comfortable moving forward** to learn more about coding. I was very nervous at first, but not anymore” (S30). These learners expressed both a desire to learn more (e.g., eager, very much interested, more interested) and a growth in confidence (e.g., confident, not [nervous] anymore, comfortable).

Theme 8: Not interested (n=4) - After taking the class, a few adult learners decided they were not interested in computer programming. For these adult learners, the class did not focus enough on motivating the importance of computer science, or they failed to learn enough to gain confidence in their ability to code. This theme emerged from a group of codes that appeared 4 times in the responses for Q3. Participants said: “Maybe. It seems important but I haven’t seen a need for it yet in my career. Also **I don’t feel I am motivated** at this time to learn it” (S29). “**Not interested** because I don’t know how to code” (S32).

**Q4. After *Code Beats*, are you interested in a career in computer science?
Why or why not?**

The responses to this question were grouped into two themes: “I can use computer programming in the future” (13) and “I’m not interested in computer programming” (4), presented in Figure 21.

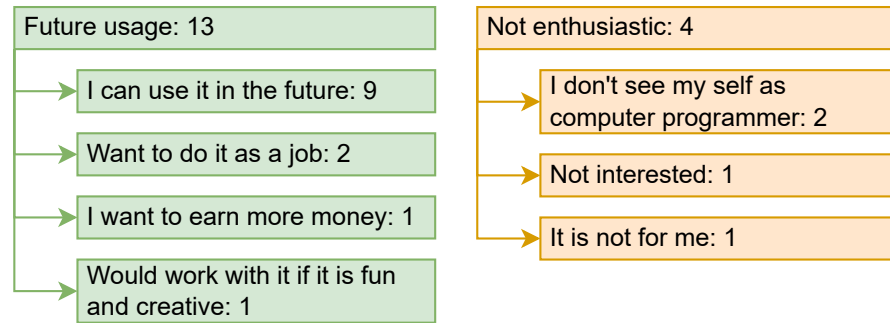


Fig. 21.: Q4 - Interested in a career?

Theme 9: Future usage (n=13) - Many adult learners believed they could and would use computer programming in the future. During completing our class, participants’ confidence in their ability to program a computer grew enough that they could imagine working in the field. This theme emerged from a group of codes that appeared 13 times in the responses for Q4. Some examples of responses that we received for this question were: “Maybe, if the coding can be used in a fun way & in a company that was creative. I am not exactly sure what the jobs would entail if I did it as a career. If I got involved with a game company, **I am sure I would love it** and learn fast” (S2). “Yes. I have a disability that makes me rely on jobs at a desk. **This could be a great fit**” (S7). “Yes. I want to do something different, **learn new technology and get paid more money**” (S10). “Yes, I love working on computers! I think that **coding is in my future** and that I am more than capable of getting the job done!” (S19). “After *Code Beats*, **I am interested in a career in**

computer science, because it has so many applications, it is rampant everywhere, and very useful. I especially like how much computers can change our lives, make things more easy for us, and do things faster than without any technology” (S22).

Theme 10: Not enthusiastic (n=4) - A few adult learners were not enthusiastic about programming. These participants’ lack of enthusiasm stemmed from a lack of interest and, in some cases, was influenced by difficulties that the adult learner had in understanding the class materials. This theme emerged from a group of codes that appeared 4 times in the responses for Q4. Participants said: “Probably not because **I don’t see myself doing anything with computer software**. I only really understand programming” (S1). “**No not interested in a career in this field**. I was so lost in this class, by trying to program music” (S32).

Q5. Did the use of music in *Code Beats* change what you thought about computer programming? If so, how?

The responses to this question were grouped into two themes: “The use of hip hop made it easier / fun to learn.” (25) and “The use of hip hop had no affect on learning” (3), presented in Figure 22.

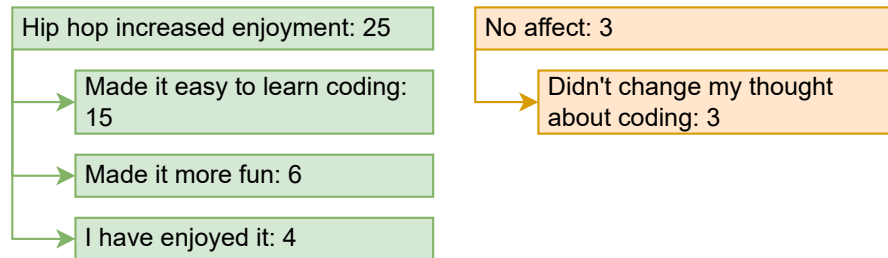


Fig. 22.: Q5 - Did the use of music change your attitude?

Theme 11: Hip hop increased enjoyment (n=25) - Almost all adult learners thought using hip-hop increased their class enjoyment. Hip-hop made the class more engaging, even for adult learners who struggled to learn the material. This theme emerged from a group of codes that appeared 25 times in the responses for Q5.

Participants said: “Yes, it made it easy. You were able to create a finished product and you could learn from your mistakes or see on the keyboard what sounded good and try to fix the code to match it. I have no music background so it was challenging, but **I enjoyed it** and it drove me to figure out how to do it better” (S2). “Definitely! **It was instant gratification.** Basically anyone could do coding the way it was taught through music with *Code Beats*. You didn’t have to be musical but if you were, it helped. With the tracks laid down already, **it was fun to write** and change the code to make different sounds and to learn coding terms...loops, variables, lists and more” (S12). “Yes it did, I thought was a **fun and interactive way** to learn coding ” (S13). “Yes, I always thought that computer programming was hard. The code itself makes no sense just by looking at it, however with *Code Beats* I understand that the coding has meaning and is formatted to give instructions to complete a task” (S19). “Absolutely. That was the best, **most fun part** (including *Dave’s* dancing). Even though I didn’t get most of the material, **it was refreshing** to see that you guys are normal and have a sense of humor. Very different from old fashioned programming in UNIX/C or FORTRAN” (S27).

Theme 12: Hip hop had no affect(n=3) - Only a few adult learners did not think the use of hip-hop affected the class. This theme emerged from a group of codes that appeared 3 times in the responses for Q5. Some examples of responses that we received for this question and were grouped in this theme are: “**No. It didn’t**” (S10). “**No, I just know the behind the scenes better!**” (S18).

5.5 Discussion

5.5.1 Perception about Computer Programming - Before *Code Beats*

Before *Code Beats*, most adult learners perceived coding as difficult, boring, or something they could never do. This is consistent with past findings; previous studies found that adult learners, prior to exposure to computer programming, thought that programming was difficult, boring, or something that they could not learn [1]. It seems that adult learners are unlikely to consider computer programming as a career option without external encouragement. Furthermore, providing an initial, positive programming experience may be crucial to whether they will ever consider studying CS.

Despite this initial reluctance, adult learners sometimes changed their minds when they realized they could learn programming by making music. Participants mentioned that *Code Beats* “seemed like a fun way to try it out” and “sounded like a cool thing to check out”, and that *Code Beats* “[was] the best fit for me” due to its focus on music. The focus on the music seemingly provided adult learners with additional motivation; they seemed to think that even though they would understand the coding, at least they would be working with music they like. Furthermore, anecdotally, it seemed to make an even greater difference for adults with musical talent. It made them more comfortable as they knew and understood the musical concepts, making the computational concepts less daunting. Overall, data suggest that associating programming with music is a promising way to attract adult learners.

5.5.2 Perception about Computer Programming - After *Code Beats*

After participation in the program, adult learners’ perception of computer programming seemed to be much more positive. They were more confident in their ability

to program, were more likely to consider furthering their programming knowledge, and some even thought that *Code Beats* made coding easy. I believe that at least part of this change in perception was because we taught computational concepts by relating them to familiar musical concepts. It seems that this technique, which led to a shift in motivation for young learners [50], may also benefit adult learners.

5.6 Limitations and Threats to Validity

The following chapter outlines the outcomes of a virtual course that was attended by participants who volunteered for the experience and had full access to it. However, it is important to note that this group may not be a representation of all adult learners, and this presents a potential external threat to the validity of the study. As a result, replicating the same study and obtaining identical results may not be feasible.

Furthermore, the surveys were limited to five questions instead of more in-depth interviews. Additionally, the survey questions were asked only after the camp. Hence, responses to questions about perceptions prior to the course demonstrate students' reflections about how they might have thought before the course.

I conducted the Reflexive Thematic Analysis with the assistance of a research associate who had knowledge of the objectives of the study. Although I followed all of the protocols outlined by the Braun & Clarke method [49], there is still a possibility that it could pose a threat to the validity of the study.

5.7 Conclusion

This chapter presented the use of music to teach the foundational concepts of computer programming to adult learners during a five weeks course. The activities in this course used actual hip-hop songs transcribed to a programming platform, where participants could modify a part of the song to apply the computer programming

concepts introduced during the classes.

By the end of the course, changes in adult learners' perception of computer programming were documented, which they had previously perceived as boring, difficult, or impossible to learn. After *Code Beats*, they started to think of computer programming as something they could learn and do, demonstrating an interest in looking for more opportunities to learn these important skills. Finally, using hip-hop to teach computer programming was found useful, motivating adult learners and making the learning process enjoyable.

CHAPTER 6

ENGAGING STUDENTS WITH NO MUSICAL BACKGROUND

While countless efforts to broaden CS education exist within high-income and developed nations [51], few efforts directly target regions outside of North America and Europe. Many of these regions, such as South America and Africa, could benefit the most from these educational initiatives, as there are large population bases that, as the cost of computers drops, are quickly starting to gain access to the necessary infrastructure to enact innovative technology education programs [52]. Therefore, the CS education community needs to prepare to serve the needs of CS education for *all*, specifically for *all* countries [53].

One clear challenge for CS education research is that approaches may not hold across cultures. Work has begun to investigate the potential effectiveness of CS education interventions in Latin America. Still, this initial work lacks rigor, so the question of which approaches will transfer cleanly remains open [54]. When focusing on Brazil, the largest country in South America, one of the more obvious challenges of transferring approaches across cultural boundaries quickly becomes clear: There are few schools with computer labs [55].

Specifically for initiatives that use music to broaden CS education, when analyzing the prior music experience of students in existing studies, it is evident that most students had some prior knowledge. While some studies do not report this data [26, 30, 29, 27, 33], the studies that do provide this information indicate that students had previous experience with music. For instance, Zhang et al. [56] indicated that 60% of

students had prior music knowledge, while Koppe [31] found that 30 to 50 percent of participants in their workshop had previous experience with music. Similarly, in the experiment reported on Chapter 4, where 64.7% of participants knew how to read music. Finally, Petrie [32] found that only 8 out of 22 participants were completely new to music and programming.

Further, most of these studies [26, 30, 29, 56, 33], including those presented in previous chapters, were held in the United States, where historically, students have had access to music classes in primary education. Outside of the United States, a few studies were conducted in Europe [27, 31] and one study in New Zealand [32]. In South America, and specifically in Brazil, to the best of our knowledge, this is the first study reporting the use of music to teach computer programming.

This chapter presents a case study using music to teach computer programming to students without previous music education. The study was conducted in Brazil, **where formal music education is not generally offered in the primary and secondary education system.** The course was offered in partnership with a public school during after-school hours. Students from 6th to 9th grades volunteered to participate in the study. The data collected through surveys, focus groups, and direct observation demonstrated that even in settings where music knowledge is limited, using music to teach coding can attract and motivate students to study computer programming. In fact, in these settings, it may be even more motivating.

6.1 Context of Work

6.1.1 Computer Science Education in Brazil

In Brazil, CS education is still in its infancy. Brazil does not yet have formal CS education at the primary and secondary levels. Except for the career preparation

integrated into the high school curriculum offered in some institutions in Brazil [57], there is only a reference guideline about implementing CS at the primary and secondary levels [58]. The National Learning Standards ¹ for primary education refer to computational thinking as a cross-cutting theme related to mathematics and not specifically to CS [59]. Fortunately, the number of studies in CS education at the primary and secondary levels is growing in Brazil, with studies in primary education receiving more attention than studies in secondary education [60]. Also, initiatives to broaden diversity in CS are in place [61].

One mapping study of computational thinking and programming in Brazil [60] indicated that games (e.g., Scratch) and robotics (e.g., Lego Robots and Arduino) are the most prevalent domains to teach computational thinking and that the least explored domain is media. Unfortunately, these efforts are held back by a lack of access to equipment in Brazilian schools, where students have access to desk computers in only 54% of schools, laptops in 35% of schools, and tablets in 15% of schools [62]. To overcome this situation, studies often try to use alternatives, such as unplugged activities [55] and more affordable equipment [63].

6.1.2 Music Education in Brazil

Music education in primary and secondary education in Brazil has been historically neglected. When it exists, it is focused primarily on singing and, in many schools, even this has ceased to exist [64]. Legislators attempted to make music education mandatory in 2008 ², yet in 2016 ³, this effort was reversed, and music became

¹Free translation of *Base Nacional Comum Curricular - BNCC*

²http://www.planalto.gov.br/ccivil_03/_ato2007-2010/2008/lei/111769.htm

³https://www.planalto.gov.br/ccivil_03/_ato2015-2018/2016/lei/113278.htm

an optional part of arts education that also contains visual arts, dance, and theatre.

Despite the laws and regulations, some challenges were faced in implementing music as a mandatory school component. For instance, align the universities' formation roadmap with the schools' curriculum [64]. Additionally, the laws and regulations are seen as vague, not specifying factors such as how many hours per week the music classes should be and a need for clear requirements of teachers' credentials, methods, and learning objectives [65].

One example of the art component diversity in primary and secondary education in Brazil is the data reported by Araújo and Lima [66], which shows how students responded when asked which activities the art teachers usually developed in the classes. Specifically, 31% of the students said drawing, 22% painting, 31% writing, and only 16% music. In summary, despite the attempts to make music part of primary and secondary education in Brazil, it is clear that music is not present in these levels of education. Therefore, students interested in music, such as music theory, singing, or learning how to play an instrument, must seek opportunities outside of the formal primary and secondary education environment.

6.2 Adapting *Code Beats* for Context

Some adaptations were necessary to implement *Code Beats* in Brazil. The first is translating all the activities and instructions into Portuguese without changing the activities' objectives and content. The translation occurred only on the instructions, not in the TunePad interface, TunePad documentation, or built-in functions (e.g., *playNote*, *rest*). Also, adaptations were needed in the curriculum, where not all concepts from the original format were taught, and, in some cases, the concept was approached in more than one class. The curriculum distribution is shown in Table 6.

Another difference from the original approach is the number of hands-on ac-

Day	Programming Concept	Music Concept
1	Sequencing	Melody
2	Sequencing	Melody
3	Variables and Constants	Musical Notes System
4	Functions (with single parameter)	Rhythm
5	Functions (with single parameter)	Rhythm
6	Lists	Chords
7	Lists	Chords
8	Repetitions (numeric controlled)	Rhythm
9	Repetitions (numeric controlled)	Rhythm
10	Modularization	Orchestration

Table 6.: Curriculum Distribution - Brazilian Edition

tivities during classes. Due to time constraints of the after-school session, in this implementation, students had one short and one long activity instead of two short and one long activities from the original format. While two short activities were prepared for this class, during the first days of the course, it became evident that this group would need significantly more time to complete activities, as students were taking much more time to understand the music requirements for the activity. Thus I opted to remove the second short activity from each day, and, on some days, I ended up applying only one activity, leaving the other for the following class, as shown in Table 7.

Another change is in the way the music content was taught. In the original context, even though connections were made between music and coding in the instructions, in all lessons, the CS instructor taught coding, and the music instructor taught music. In the implementation reported in this chapter, it was impossible to have the music instructor in every day of class, so the coding instructor (having witnessed the music instruction many times as part of other classes) gave the music lesson on days the music instructor was absent.

Day	Type	Link
1	Short	https://tunepad.com/project/41875
2	Long	https://tunepad.com/project/41876
3	Short	https://tunepad.com/project/41917
3	Long	https://tunepad.com/project/41918
4	Short	https://tunepad.com/project/42220
5	Long	https://tunepad.com/project/42221
6	Short	https://tunepad.com/project/42310
7	Long	https://tunepad.com/project/42311
8	Short	https://tunepad.com/project/42544
8	Long	https://tunepad.com/project/42545
9	Short	https://tunepad.com/project/42546
9	Long	https://tunepad.com/project/42839
10	Project	https://tunepad.com/project/55281

Table 7.: List Code Beats of Activities - Brazilian Edition

Finally, when implemented in the United States, students composed their own beats (from scratch) at the end of the course to participate in a beat contest. When implemented in Brazil, students did not engage with the beats contest. There were two main reasons for this. First, not all students had computer access outside of class time, and second, students' lack of previous music experience limited their ability to progress to the *create* phase of the Use-Modify-Create framework.

6.3 Research Question

This study aims to test the feasibility and understand what motivates a population without previous music experience to attend a coding course that uses music. Thus, the research question being investigated is: What motivated students to attend, continue to attend, and engage in this course?

6.4 Methods

6.4.1 Study Design

The *Code Beats* classes were organized in partnership with a public school that allowed the researchers to advertise the course for the students and use the school computer lab. The students were divided into two groups due to the limited size of the available computer lab. Both groups had the same content in the same order. Each group consisted of ten classes, over two weeks, with an hour-long class each weekday after school. The classes included computer programming lessons, music theory lessons, hands-on activities, and interactive quizzes. A typical day of classes had two hands-on activities based on an actual hip-hop song transcribed to TunePad. Students had to do their activities in one TunePad cell, which would complement the transcribed song. Students were asked to complete a voluntary pre and post-course survey at the beginning and end of the course. On the last day of the course, students were invited to participate in a focus group session.

6.4.2 Participant Demographics

A total of 55 students participated in at least one day of classes. From this total, 25 students answered the pre and post-course surveys. They were, on average, 13.5 years old (min 12 - max 15). 44% of the students self-identified as girls, 40% as boys, and 16% of the students preferred not to say. 60% of the students self-identified as White, 36% as Mixed Races⁴, and 4% as Black. 52% of the students said they had computer classes in the past. However, for 60% of the students, this course was their first experience with computer programming, and 32% had less than one year

⁴Translation of *Pardo*, multi-racial Brazilians.

of experience in computer programming. 80% of the students did not play a musical instrument, and 88% said they had never had music classes. The students' preferred musical genres are funk (52%), Brazilian country music (44%), pop (40%), rap (40%), rock (36%), and hip-hop (32%).

6.4.3 Data Collection and Analysis

A pre and post-course survey was administered to document the changes between pre and post-course and to provide data to answer the research question. The pre and post-course survey is the same one used in previous experiences of *Code Beats*, and it is adapted from Mouza et al. [67], inspired by Ericson and McKlin [68]. The survey includes 26 questions and asks students to rate their agreement or disagreement using a Likert scale format, from 1 - "Strongly Disagree" to 5 - "Strongly Agree." The questions were grouped into four categories, (a) confidence in CS, with 16 questions (e.g., "I have a lot of self-confidence when it comes to computing."); (b) belonging to CS, with 3 questions (e.g., "I feel like I "belong" in computer science."); (c) gender equality in CS, with 3 questions (e.g., "Girls can do just as well as boys in computing."); and (d) students' future in CS, with 4 questions (e.g., "Someday, I would like to have a career in computing.").

Survey data on students' attitudes toward computing were scored, entered into a spreadsheet, and subsequently exported into statistical software, where means and standard deviations were calculated to assess changes from pre to post-administration. To test the statistical significance of the difference between pre and post-course surveys, the Two Sample t-test was performed when the data were normally distributed and the Wilcoxon Rank Sum test when the data were non-normally distributed. To test the normal distribution, the Shapiro-Wilk test was used.

In addition to survey data, focus groups were also conducted with 24 students

to assess student experience. Focus group questions addressed student motivation for attending the course (e.g., “What made you decide to attend this course?”); experience with coding (e.g., “Tell me about your experience coding in TunePad. How did it work?”); potential surprises (e.g., “Did anything surprise you about the course?”); and the role of music in sustaining interest (e.g., “Did the fact that you were making music make you more or less interested in this course?”). Focus group data were analyzed qualitatively to identify emerging themes.

All questions in the survey and in the focus group were in Portuguese, the students’ native language, and the answers were translated into English to perform the analysis and report the results.

The author of this dissertation acted as an instructor in the course, helped by another instructor with an art degree with a concentration in music. He teaches music and art classes in Brazil. To corroborate the findings, their observations during the classes and the main challenges faced during the course are reported.

6.5 Results

6.5.1 Students’ Motivation

High prior interest:

Student scores on a pre and post-survey were analyzed, and it was found that they started the class with unusually high scores for all measures, and these scores did not change much throughout the session.

Table 8 reports each category’s pre and post-course responses. The first column has the category. The second and third columns contain the average of responses for pre and post-course surveys, with their standard deviation in parenthesis. The fourth column shows the difference between the pre and post-course survey averages.

Category	Pre	Post	Diff	p-value
Confidence in CS	4.0 (1.2)	4.2 (1.0)	0.2	0.0627
Belonging in CS	3.0 (1.3)	3.3 (1.3)	0.3	0.0756
Gender Equality	4.7 (0.8)	4.7 (0.8)	0.0	0.1198
Future in CS	3.6 (1.3)	3.7 (1.2)	0.1	0.5999

Table 8.: Pre and Post-course Survey

The last column has the p-value for each category. For all but “Gender Equality,” the statistical significance was tested using the Two sample t-test. For the category “Gender Equality,” the test performed was the Wilcoxon test. The difference between the pre and post-course survey averages is not statistically significant for any category.

Motivation for attendance:

This after-school course was entirely voluntary. However, it required students to sign up ahead of time, and each day, students had to return to school, as it was held after they had already returned home. When students were asked what made them decide to attend this course, many students cited an interest in learning music. For instance, S18 answered, *“To fill my free time and improve my music theory knowledge.”* S29 stated, *“Because I thought it would be cool, and I always wanted to learn about music.”* Finally, S35 answered, *“My interest in music. Learn about music theory.”* An interest in learning CS was also a factor. For example, S26 answered, *“Because I’m interested in computer science, and I guess it is cool.”* S16 stated, *“Because computer science is one of my interests.”* And S31 answered, *“I am interested in computers, and it appeared to be an opportunity to start studying them. The music made it more fun.”* S04 answered, *“I was interested in learning more about computer science and music.”* demonstrating interest in music and CS.

Students also attended because they thought it would be interesting and fun. For instance, S08 said, *“I decided to attend this course because it was interesting, and my friends would attend too.”* that is in line with the answer from S06, *“I decided to attend because it appeared to be fun.”* Some students indicated that it was a good use of their free time. For example, S27 answered, *“I thought it would be fun and something to do with my free time.”* S20 answered, *“I thought it was fun and I would have something to do with my free time.”*

In contrast, parents’ choice was one of the main reasons students joined the *Code Beats* in North America. For instance, in the camp held in 2021, the most common reason for attending the camp, selected by almost 70% of respondents, was “my parent or guardian recommended it.” In line with that, this reason was selected by 56.3% of respondents in the camp held in 2022. Some examples of students’ answers given during the focus group are *“my mom just signed me for this.”* and *“my mom recommended me for it.”*

Programming experience:

Students were asked to talk about their coding experience during the course. Some students said that it was easy, like S24, who said, *“It was really easy.”* One student (S09) who had experience with another programming language said, *“Easy, there is no semicolon.”* Another student with previous programming experience said *“It was not hard. The syntax was easy to memorize. The hard part was to know how to use the music. It is easier than Scratch.”* On the other hand, some students said it was hard at the beginning of the course, but it became easy with time. For instance, student S14 said, *“In the beginning was hard, and then it started to be easier.”* Likewise, student S29 stated, *“In the beginning, it was a little confusing, but now it is easy.”* Additionally, student S10 explained what helped him in the course, *“It was a little difficult, but with help from the instructors, it became easier.”* Finally,

student S36 pointed out that it was hard to memorize the commands, *“It was a little complicated. It was hard to memorize everything. With time, we could get it.”*

Unexpected pairing:

Students were asked if anything surprised them during the course. Some students said they were surprised that they could create music with coding. For instance, student S12 answered, *“The TunePad. I never imagined that it was possible to create music like that.”* Student S04 stated, *“The fact that it is possible to create music with coding.”* Another student, S06, said, *“When we created the song, even the minimal one, I was impressed.”* Additionally, one student was surprised by how the different tracks combined to make a complete song. S32 stated, *“Use all the code together. It is really interesting. I had no idea it was possible.”* Finally, one student was surprised by the music instructor’s ability. S31 said, *“The music instructor’s ability. It is possible to see that he knows what he is doing.”*

Music as a ‘hook’:

Students were asked if making music made them more or less interested in this course. The vast majority pointed out that the music was crucial to their interest, motivating them to attend the course. For instance, student S01 said, *“Yes, I would not attend with no music. When the course was advertised, you ⁵ started talking about computer programming, and it did not bring my attention, but when you talked about music, I thought, yes, this I would like to do.”* Students S12 and S18 stated, *“If there were no music, I would not be interested in / I would not attend the course.”* Some students agreed that the music made the course more interesting, but would attend as well without music. For example, student S06 answered, *“If there were no music, I would attend, but the interest would be lower.”* Student S31 stated, *“I would have*

⁵ The researcher that advertised the course.

been interested without music, but the music made it more fun.”

6.5.2 Direct Observation

Students’ interest:

The first unusual observation happened before the course started, during recruitment, when the author of this dissertation visited each class to describe the course to students briefly. The students’ reaction was positive, with a strong appreciation of the idea of using music to teach coding. Registration forms were only given to students that explicitly asked for them. 150 students asked for this form, and 55 returned them signed by their parents, registering the student and giving the research consent.

The students remained engaged and motivated during classes, paying careful attention to the music and coding lessons. After each day’s lessons, we held an interactive quiz. The students showed interest and engagement, especially during these quizzes, where they competed amongst themselves to see who could answer the most questions correctly and quickly. During the focus group interviews, when students talked about what they liked most, they consistently said quizzes.

The difference between the students who attended at least one day of classes and those who answered the pre and post-course surveys is due to students who came in the vast majority in the first one or two days of the course and did not return for classes. Additionally, even those students that answered the pre and post-camp surveys missed some classes. Following up with students and schools’ responsible, the main reason for them to miss classes was due to weather conditions, where on rainy days, it is common for students to miss even regular classes at this school.

Specifically, regarding motivation and interest during classes, students paid attention to the coding and music lessons. Of course, some students were distracted,

talking to their peers or even browsing the Internet. Still, the vast majority paid attention to the instructors and asked questions to clarify their understanding.

Musical competence and confidence:

For most students, this course was their first contact with music theory; these students did not have the basic knowledge of music, for instance, musical notes and rhythm. Even though the activities do not require deep knowledge, students had to gain at least a basic understanding to complete the activities successfully. Fortunately, students seemed to be able to pick up musical concepts quickly, and they were engaged when asked to work on their activities. They did especially well with the shorter activities, with more straightforward solutions, with an example to follow and hints that guided them to the correct answer. On the other hand, students struggled with longer, more open-ended activities, seemingly due to a lack of music knowledge or perhaps confidence. When the instructor solved these longer activities with the class, students could answer specific questions about what commands to use and how to use them but struggled to do this alone. Similarly, when students were asked to create their own compositions on the last day of class, they struggled to do so. It seems that their lack of confidence in their own musical skill made them hesitate or even stop when asked to make musical choices, even though they had a good understanding of the necessary programming concepts and commands to implement a solution.

On the last day of the course, while some students were participating in the focus group, the rest were composing the class song. The initial idea was to have individual students composition. However, as students struggled to solve the long activities during the classes, the researchers decided to have one single composition for the group of students. During this process, students could indicate the commands to use in TunePad with its parameters but needed help deciding the musical composition. So the music instructor conducted that.

6.6 Limitation and Threats to Validity

This study was performed in specific settings. Therefore, its results may not be generalizable in other contexts with populations from different backgrounds. The number of students attending the course differs from those who answered the pre and post-course surveys and participated in the focus group. Unfortunately, it was impossible to collect the reason for drop-out from students that did so. The interviews and data analysis was conducted by the same researcher who developed and delivered the *Code Beats* program. Therefore, it may represent author bias.

6.7 Conclusion and Future Work

This chapter reports a case study of teaching computer programming with music to students without music experience in a country where, predominantly, music is not part of students' formal education. Based on an approach already tested and validated in a context where most students have previous music knowledge, the original curriculum and activities were adapted to this new setting. It was impossible to see a statistically significant difference between the pre and post-course surveys regarding students' confidence in CS, CS belongingness, gender equality in CS, and students' future in CS. However, the pre-course survey answers were already high for these categories. According to students' interviews and researchers' observations, the use of music was a key factor in attracting and engaging students in this course, with some students saying that they would not be interested in attending the classes without music.

This case study shows that it is possible to use music to attract students in a setting where music is not part of their formal education, adding new findings to the growing body of CS education research that looks for ways to attract, motivate and

engage students into this area.

New musical genres can be tested in future work to see if they produce different results. Additionally, a more extensive curriculum with more classes can be tested to cover all the foundational computer programming and music concepts.

CHAPTER 7

STUDY OF SCAFFOLD-BASED ACTIVITIES FOR MUSIC CODING

One way to keep students engaged and improve their confidence in learning computer programming is by decreasing the friction of initial computer programming instructions and exercises. Studies have shown that scaffolding can reduce the possibility of overwhelming and frustrating students new to programming, especially during independent learning activities [69].

One way to do that is by implementing scaffold-based activities [70]. Although there is an extensive body of literature on using scaffold-based activities in programming broadly [71, 72, 73], their use in a music coding context can require non-trivial adaptations. For example, traditional programming activities often require a particular state to be attained at the end of an activity, such as sorting a given list or successfully navigating a maze. Because music is an ephemeral art form, producing only a series of sounds over time, it can be challenging to create the same types of activities in the domain of music, and the consequences of adapting these scaffolds to a music coding context have not yet been studied.

This chapter describes a study implementing scaffolding within a two-week virtual summer camp using the *Code Beats* approach. Before this camp, I adapted three common scaffolded activities—complete the code, buggy code, and reorder the code—to the domain of music coding, subtly offering musical support while still requiring students to learn and use programming concepts. During this camp, I examined these different scaffold-based activity types applied to in-class activities and compared them

in terms of perceived difficulty and solution correctness.

7.1 Related Work

7.1.1 Scaffold-Based Curricula and Activities

The term scaffolding has been used to refer to the process where a teacher or a more knowledgeable peer assists a learner, altering the learning task so the learner can solve problems that would otherwise be out of reach [74]. One of the benefits of scaffolding is increasing learning engagement and motivation, permitting students to work with artifacts that are complex enough to be meaningful and fun [75].

Scaffolding can be used as a teaching and learning strategy in terms of curriculum, for instance, the Use-Modify-Create framework [40] and PRIMM [76], and in directly scaffolded activities, such as worked example, buggy code and Parsons Problems [77].

When it comes to activities, worked examples, incomplete code, buggy code, modifying code, and parsons problems are listed as activity types that are scaffold-based and can be used in a wide range of ways to teach computer programming concepts [78, 77].

One type of scaffold-based activity, Parsons Problems, was initially described as “Parson’s Programming Puzzles” [71]; the original idea resembles a drag-and-drop puzzle where all code fragments must be ordered to form complete working code. To date, there are several variations in Parson’s problems. For instance, there are specific platforms that allow users to solve Parsons Problems [79], there are Parsons Problems implemented in ebooks [80], and, perhaps most naturally, Parsons Problems implemented in block-based environments [81].

Another common scaffold-based activity is intentionally placing bugs within learning activities, asking students to explain or fix bugs carefully designed to ad-

dress key concepts and highlight common errors [72, 82, 83]. A final scaffold-based activity type is to ask students to complete partial programs, known as completion strategy or the completion problem [73, 84].

7.1.2 Scaffolding Music-based Programming

Despite the good results in motivating and engaging students in computer programming classes using coding music [26, 29, 33], none of the approaches related here deeply apply scaffold-based techniques in the activities.

To our knowledge, the only work that applies scaffolding techniques to music coding is the one reported in previous chapters of this dissertation, where the curriculum and how the activities are distributed in classes are based on the Use-Modify-Create framework [40]. In this approach, students were given relatively lightly scaffolded exercises. For example, they were given backing tracks and asked to add a new track, which added a melody or a percussion instrument, and the new track only contained light guidance.

The music domain presents unique challenges to scaffolding techniques such as complete the code, buggy code, and Parsons Problems. For instance, any activity based on reordering statements can become more challenging in a musical context, as the order of notes has a profound effect on the melody, and notes should match the underlying chord progression, which adds complexity. In addition, any activity based on fixing bugs, filling in missing statements, or adding commands is subject to the same underlying chord progression, as well as the key of the song, and asking students to add notes without strong guidance can lead to melodies that violate musical guidelines, thus sounding off to the ear. It is clear that there are several new challenges when using scaffold-based programming activities in the musical domain.

7.2 Adapting Scaffolding for Music Coding

In this chapter, I applied scaffold-based techniques that are normally used to teach programming, such as complete the code, debug the code, and reorder the code, and re-purposed them for implementation in a musical setting. First, I introduce them and explain the adaptations needed to use them in a musical setting. For simplicity, the term “scaffold-based activity type” will be referred to as “activity type” in this chapter.

7.2.1 Complete the Code

This activity type, inspired by program completion [85], requires students to fill in the blank, both musically and in code. From a coding perspective, the student is given an almost complete code snippet that is not compiling due to the fact that it is incomplete. To complete the code, the student has to define a variable, add a parameter, or add commands. From a musical perspective, students are given an entire backing track and an almost complete melody, and their task is to add one or more notes to complete the melody.

Listing 7.1: Complete the Code - Example 1

```
1  """
2  1. Complete the constant declaration for A with the right value (
    MIDI number) .
3  """
4
5  G = 67
6  A = ?
7  Bb = 70
8
9  playNote(G)
10 playNote(G)
11 playNote(A)
12 playNote(Bb)
```

Listing 7.1 is an example of this activity type, where students have to define the variable A to complete the melody.¹

Listing 7.2: Complete the Code - Example 2

```
1  """
2  1. Complete the list declaration with the following constants. D,
   C, B, B
3  2. Complete command for with a variable that will be used on
   command playNote
4  """
5
6  D = 74
7  C = 72
8  B = 71
9  notes = [ ]
10
11 for ? in notes:
12     playNote(note, beats = 2)
```

Listing 7.2 shows another, more complex example of this activity type, where students have to (1) complete the list declaration by adding four notes and (2) replace the question mark with the name of the variable that controls the loop.²

7.2.2 Buggy Code

For this activity type, inspired by finding and fixing errors [86], students are given code that contains no syntax errors, but does contain one or more semantic bugs. In the context of music, a semantic bug is an incorrect note in a melody or harmony, an incorrect rhythm, or a combination of the two. To fix these, students have to identify the bug and subsequently modify the code. As with complete the code, they can identify the bug by listening to the music. Their prior exposure to

¹<https://youtu.be/WfK09YjwSPc> shows a simulation of a student completing this activity in TunePad, as described above.

²<https://youtu.be/XyXPYJrJrvs> shows a simulation of a student completing a more complex “Complete the Code” activity.

Western music and hip-hop specifically will guide them toward finding the incorrect note (or rhythm).

Listing 7.3: Buggy Code

```
1  """
2  1. One constant (i.e., note) has the wrong MIDI value. Find it
   and change it.
3  """
4
5  G = 67
6  A = 45
7  Bb = 70
8
9  playNote(G)
10 playNote(G)
11 playNote(A)
12 playNote(Bb)
```

Listing 7.3 shows an example of this activity type. Students must play the song and identify which note is incorrect, trace that back to the incorrect MIDI number, and then change the MIDI to the correct value.³ As you can see, this example is intentionally almost exactly the same as Listing 7.1, from complete the code, the difference being that in the buggy code example, students have to identify which variable is incorrect, whereas, in the complete the code example, it is clear which variable needs to be defined.

7.2.3 Reorder the Code

This activity type is inspired by Parsons Problems [71], where the student receives a complete code listing, yet statements within that listing are ordered incorrectly. Students’ objective in these problems is to identify which lines are out of order and reorder them correctly, ultimately building a working program and song.

³<https://youtu.be/qTrn4g51vzk> is an example of how students might solve a “Buggy Code” activity.

Listing 7.4: Reorder the Code

```
1  """
2  1. Reorder the commands in this track to create a melody for this
   beat.
3  """
4
5  notes = [ D, C, B, B ]
6  D = 74
7  C = 72
8  B = 71
9
10     playNote(note, beats = 2)
11 for note in notes:
```

At Listing 7.4, there is one example of this activity type where students have to identify the code that is out of order and reorder it.⁴ In this case, students should (1) understand how a “for loop” repeats commands that follow it (not precede it) and (2) that a list only can use variables or constants after they are declared.

7.3 Research Questions

This research aims to compare the use of different types of activities in terms of students’ perceived difficulty and task performance (i.e., correctness). The research questions are:

RQ1: How do students’ perceptions of task difficulty vary among different activity types?

RQ2: Which activity type leads to better performance (i.e., task correctness) among students?

⁴<https://youtu.be/0IND90AH0Aw> is an example of how a student might solve a “Reorder the Code” activity.

7.4 Methods

7.4.1 Participant Demographics

This *Code Beats* summer camp was conducted online, with two groups of students, one in the morning (Camp A) and another in the afternoon (Camp B), to accommodate student interest and demand, since the students chose the section they would like to attend, the participants were not randomly assigned. The average number of unique online viewers was 89.2 for Camp A and 64.1 for Camp B. A total of 87 students completed at least one in-class activity for Camp A and a total of 62 for Camp B.

If students submitted at least one in-class activity, I could match their usernames with the brief pre-camp survey, which collected demographic information. It represents more than 80% of the total. For Camp A, 73 students met these conditions, and that population was, on average, 11.7 years old (min: 9 - max: 14). The self-reported gender distribution was 43.8% girls, 50.7% boys, and 5.5% prefer not to say. The self-declared race distribution was 28.8% African-American; 19.2% Asian; 2.7% Hispanic; 6.8% Multiracial, not Hispanic; 2.7% Other; 4.1% Prefer not to say; and 35.6% White. For Camp B, 50 students met these conditions, and that population was, on average, 11.9 years old (min: 7 - max: 15). The self-reported gender distribution was 34% female, 64% male, and 2% prefer not to say. The self-declared race distribution was 24% African-American; 14% Asian; 8% Hispanic; 6% Multiracial, not Hispanic; 10% Other; 6% Prefer not to say; and 32% White.

Additionally, in terms of music knowledge, 38.4% of students from Camp A declared that they did not know how to read music and did not play a musical instrument. For Camp B, 22% of the students declared that they do not know how to read music and neither plays a musical instrument. Concerning music genre preference,

39.7% of students from Camp A stated that hip-hop is their preferred music genre. For Camp B, this is true for 44% of the students.

7.4.2 Study Design

Since two different camp sessions were held, it was possible to examine and compare across activity types, as the same music coding activity was assigned, with minimal changes, using a different activity type for each camp.

Day	Programming Concept	Camp A	Camp B
1	Sequencing	Complete	Buggy
2	Variables and Constants	Buggy	Complete
3	Functions (single parameter)	Complete	Buggy
4	Functions (multiple parameters)	Buggy	Complete
5	Lists	Reorder	Complete
6	Repetitions (numeric control)	Complete	Reorder
7	Repetitions (list iteration)	Reorder	Complete
8	Repetitions (nested lists)	Complete	Reorder

Table 9.: Programming Concepts and Activity Types

As shown in Table 9 for Day 1 - Camp A, the “Complete the Code” activity type was used, which was possible to compare directly with the “Buggy Code” activity type used in Day 1 - Camp B because the music coding activities were otherwise identical. Note that each day students completed two activities, meaning that I have a total of 8 music coding activities with which to compare “Complete the Code” with “Buggy Code” (Days 1-4, 2 activities each day), and similarly eight activities with which to compare “Complete the Code” with “Reorder the Code” (Days 5-8, 2

activities per day). This allowed me to examine each activity type in the context of completing the same music coding activity ⁵.

To measure students' perception of the difficulty of each activity type, an interactive polling system (Mentimeter) was used during the live class, which asked students to respond to the following statement: *I found the activity...* with one of the following choices: *a) too easy; b) about the right level for me; or c) too difficult*. To measure the correctness of each student's solutions, the source code each student submitted during the camps for each given music coding activity was also collected by downloading them directly from the programming platform after the camp.

7.4.3 Data Analysis

To compute the perception of difficulty for each activity type, I counted the answers collected via the live polls for each activity type that was used.

To verify the correctness of each activity, each student's solution was analyzed and compared with an expected solution. As a result, each solution was classified as "Correct" or "Incorrect". A solution was only marked as "Correct" if that solution was 100% accurate in both coding and musical parts. For instance, the solution must use the proper programming constructs and the expected musical instruments and notes in the expected order. Using Listing 7.2, as an example, the solution is only classified as "Correct" if the order of the musical notes in the list definition is exactly as expected (`notes = [D, C, B, B]`) and the repetition variable is defined correctly (`for note in notes:`). After identifying whether an activity was correct or incorrect, results were analyzed for each in-class activity on each day of camps A and B and counted for each activity type.

⁵The list of activities can be accessed here: <http://bit.ly/3VwQJQn>

It is essential to state that it is possible to classify the solutions for in-class activities used in *Code Beats* as “Correct” or “Incorrect” using the criteria of the expected solution. For after-class activities and the camp project that are not within the scope of this study, the students have space to show their creativity, where a single “correct” solution is not expected for the problems.

To test the association between an activity type and level of difficulty, and activity type and correctness of the solution, I used the Chi-squared (χ^2) test for independence, with the value of alpha equal to 0.05. The Chi-squared test for independence is used to determine whether there is a statistical association between two categorical attributes [41].

7.5 Results

7.5.1 RQ1: Difficulty

Comparing the first four days of camp, where the activity types used were “Complete the Code” (8 activities, 432 responses; 54 on average, SD = 11.6) and “Buggy Code” (8 activities, 437 responses; 54.6 on average, SD = 10.1), the overall percentage of students who classified the activity as “too difficult” for “Complete the Code” is 12.0% and for “Buggy Code” is 19.5%. The percentage of students who classified the activity as “too easy” for “Complete the Code” is 20.1% and for “Buggy Code” is 17.2%. This difference was significant ($\chi^2_{(2)} = 9.26$, p-value = 0.01).

In Figure 23, I report the variation in the percentage of perceived difficulty attributed by the students for each in-class activity in the first four days of the camp. It is possible to observe that the mean percentage of students that rated the activity as “too difficult” for “Complete the Code” activities is lower than for “Buggy Code” activities. At the same time, the mean percentage of students who rated the activity

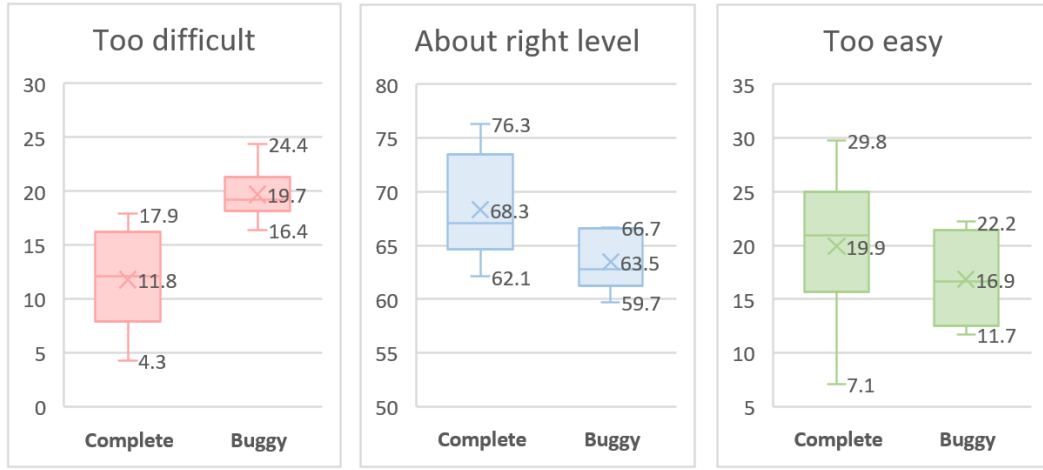


Fig. 23.: Difficulty - Results for Complete the Code and Buggy Code

as “about right level for me” is higher for “Complete the Code” than for “Buggy Code”. Finally, the mean percentage of students who rated the activity as “too easy” is higher for “Complete the Code” than for “Buggy Code”.



In the context of coding music, “Buggy Code” is perceived as more difficult than “Complete the Code”.

Comparing the last four days of camp, where the activity types used were “Complete the Code” (8 activities, 308 responses; 38.5 on average, SD = 6.7) and “Reorder the Code” (8 activities, 312 responses; 39.0 on average, SD = 7.6), the overall percentage of students classifying the activity as “too difficult” for “Complete the Code” is 18.5% and for “Reorder the Code” is 34.0%. The percentage of students classifying the activity as “too easy” for “Complete the Code” is 12.7% and for “Reorder the Code” is 13.1%. This difference was significant ($\chi^2_{(2)} = 20.61$, p-value = 0.00003).

In Figure 24, I report the variation in the percentage of perceived difficulty attributed by the students for each in-class activity in the last four days of the camp. It is possible to observe that the mean percentage of students that rated the activity

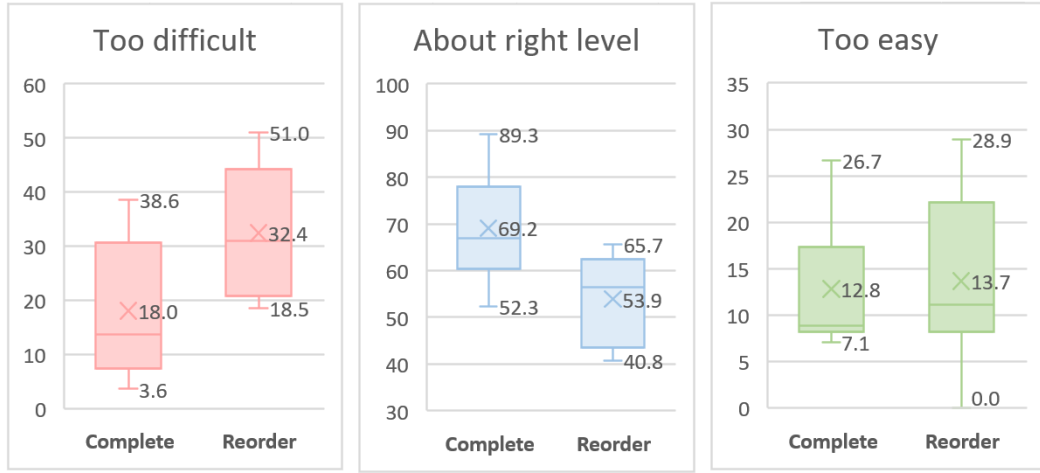


Fig. 24.: Difficulty - Results for Complete the Code and Reorder the Code

as “too difficult” for “Complete the Code” activities is lower than for “Reorder Code” activities. Additionally, the mean percentage of students who rated the activity as “about right level for me” is higher for “Complete the Code” than for “Reorder the Code”. Finally, the mean percentage of students who rated the activity as “too easy” is slightly higher for “Reorder the Code” than for “Complete the Code”.



When coding music, “Reorder the Code” is perceived as more difficult than “Complete the Code”.

7.5.2 RQ2: Correctness

Comparing the first four days of camp, where the activity types used were “Complete the Code” (8 activities, 355 coded solutions; 44.4 on average, SD = 10.3) and “Buggy Code” (8 activities, 358 coded solutions; 44.8 on average, SD = 9.6), the overall percentage of students with a correct solution using “Complete the Code” as the activity type was 51.5%, and using “Buggy Code” was 62.0%. The overall percentage of students with an incorrect solution using “Complete the Code” as the activity type was 48.5%, and using “Buggy Code” was 38.0%. Considering the group

of answers, the $\chi^2 - test$ was significant ($\chi^2_{(1)} = 7.95$, p-value = 0.005).

This also can be observed when we look at the variation in the percentage of correct and incorrect solutions for all in-class activities for the first four days of the camp. For example, in Figure 25, we can see that the mean percentage of incorrect solutions using “Complete the Code” is higher than that of solutions for activities using “Buggy Code”. Coexisting, the mean percentage of correct solutions is higher for activities using “Buggy Code” than solutions using “Complete the Code”.

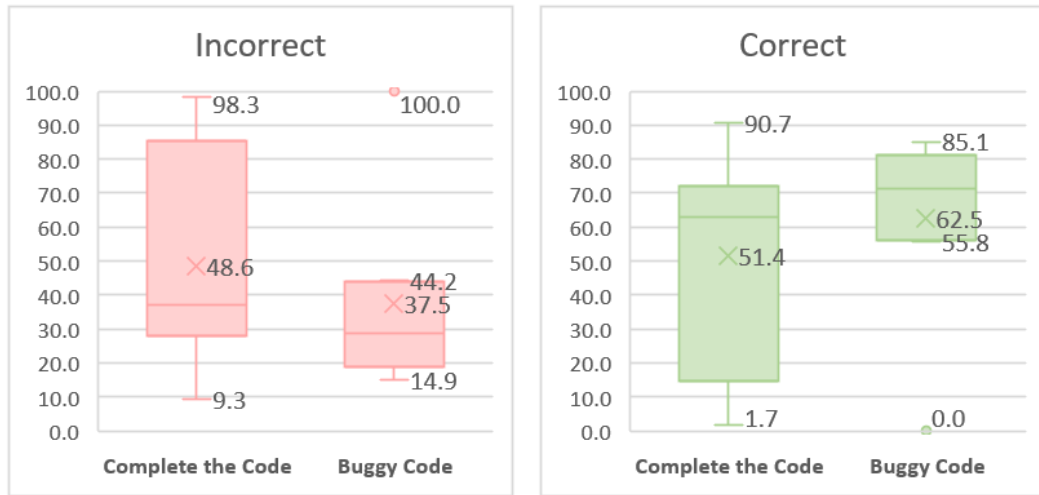


Fig. 25.: Correctness - Results for Complete the Code and Buggy Code



When coding music, students perform better when the activity was scaffold-based on “Buggy Code” than “Complete the Code”.

Comparing the last four days of camp, where the activity types used were “Complete the Code” (8 activities, 279 coded solutions; 34.9 on average, SD = 9.1) and “Reorder the Code” (8 activities, 263 coded solutions; 32.9 on average, SD = 8.8), the overall percentage of students who reached the correct solution for the activity that used “Complete the Code” as the activity type is 37.3%, and for “Reorder the Code” is 40.3%. The overall percentage of students who did not reach the correct solution

for the activity that used “Complete the Code” as the activity type is 62.7%, and for “Reorder the Code” is 59.7%. Considering the group of answers, the χ^2 – test was not significant ($\chi^2_{(1)} = 0.52$, p-value = 0.47), as students seemed to struggle equally on both activity types. This also can be observed when we look at the variation in the percentage of correct and incorrect solutions for all in-class activities for the last four days of the camp in Figure 26.

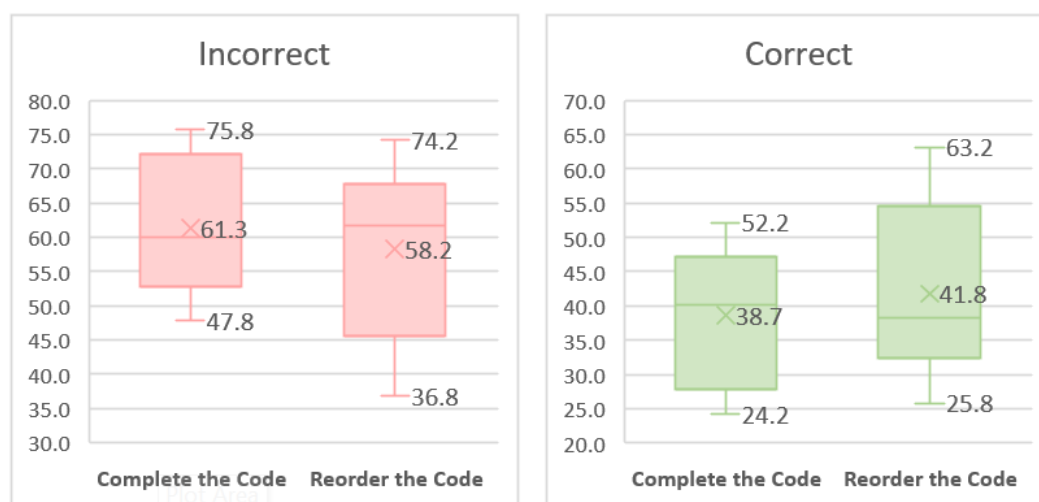


Fig. 26.: Correctness - Results for Complete the Code and Reorder the Code

To summarize, 51.5%, 62%, 37.3%, and 40.3% of students provided correct answers for the four combinations of activity types and time periods, even though the activities were designed to be easy to complete correctly. There is clearly room for improvement when, in all cases, more than a third of students were struggling to complete activities correctly.



When coding music, students struggled to create correct solutions, even with scaffold-based support.

In the context of this study, based on students’ classification, I found that “Complete the Code” is perceived as less difficult than “Buggy Code” and “Reorder the

Code”. At the same time, students performed better in terms of correctness for “Buggy Code” activities than “Complete the Code”, possibly because correctness in music (i.e., it sounds good) differs from correctness on programming assignments, which require a near-exact match. Especially during the second week of classes, less than 50% of the students could complete the activities correctly. While there are some potential explanations—students do not always read instructions carefully—I believe that student difficulties with these activities were deeper than this.

7.6 Limitation and Threats to Validity

The virtual format of this camp made the students’ participation in the activities hard to influence. Consequently, the number of students who attempted to complete the activities was lower than those who watched the class. Additionally, the number of students I could analyze the activity solution is lower than those who rated the activity’s difficulty. This difference may be because some students were not logged in to the coding platform to perform the activities or were using credentials different from the one informed. This may skew the data, especially the analysis of students’ final beats, towards the most engaged students. However, that data should be seen as the best-case data and may change for in-person classes, where a more significant percentage of the class would participate actively in activities.

Another possible threat to validity is the percentage of the students with previous knowledge of music, as those who answered that they know how to read music or play a musical instrument was high (Camp A - 61.6% and Camp B - 78%). Unfortunately, this percentage might not be possible to reproduce in future classes; it is unclear whether the high percentages were because musical people were attracted to this class or whether this would hold for most populations. It is important to mention that the musical genre used in *Code Beats* was not the preferred musical genre for

the majority of the students.

7.7 Conclusion

As the main contribution to the CS Education community, this study compares three different scaffold-based techniques in terms of perceived difficulty and correctness that are applied to a specific approach, teaching coding using music, but can also be used as a starting point for comparison in different contexts and approaches.

For *Code Beats*, which aims to attract students that would have their first contact with computer programming, I believe it is of the utmost importance that students do not perceive activities as difficult, which could lead to frustration and class abandonment.

CHAPTER 8

USING DOMAIN-SPECIFIC, IMMEDIATE FEEDBACK IN *CODE BEATS* TO SUPPORT STUDENTS

Motivated by the results from the experiment that uses scaffold-based activities for music coding, presented in Chapter 7, I decided to implement and evaluate an extra level of help to the students in the learning process, using domain-specific, immediate feedback to support students during the activities-solving process.

While *Code Beats* has shown overall promise, as with any external domain, there is a significant caveat that students must understand two domains at once, computer science and music. To avoid unnecessary barriers to learning, researchers and instructors must pay careful attention to this secondary domain, ensuring that either the students already have the necessary background or that they are given this background. Creating positive experiences (including enjoyment and growing confidence) is particularly important because they profoundly impact students' intention to persist in computing [87].

Learning programming can be challenging, and most students need help to make progress. Therefore, providing timely feedback is an important factor in learning [88] for improving knowledge and acquiring skills [89]. Specifically, immediate automated hints can help students progress in their learning by providing instant and relevant feedback to correct their mistakes and point them in the right direction to advance through activities [90]. Some studies use feedback with programming activities, which has been shown to be helpful [88, 89]. However, there is still a need for further research

on designing programming feedback to create positive, motivating, and engaging programming experiences while promoting performance and learning [91]. Furthermore, the feedback necessary to ensure good musical choices is orthogonal to the feedback needed to create correct programs; a program can be syntactically valid and even play the right number of notes but still be woefully out-of-key.

This chapter presents the results of a quasi-experimental study using the *Code Beats* approach. Students from one group received expert-authored, domain-specific feedback in their in-class activities to help them during the solution process. In comparison, students from another group that received the exact instructions had to solve the same activities without integrated feedback. The data collected during eight classes, and twenty-four activities, shows statistically significant evidence that the group that received feedback completed their activities with higher scores of correctness. In addition, focus groups conducted after the camp showed that the students valued the feedback and that they increased student confidence.

8.1 Related Work

8.1.1 Feedback to Improve Code Learning

Formative feedback is defined as information communicated to the learner intended to modify their thinking or behavior to improve learning [92]. An effective feedback is defined as non-evaluative, supportive [92], timely, specific [93, 92], positive, and corrective [93].

The timeliness of feedback can be divided into three categories. First, immediate feedback is often more effective on complex tasks when students have less prior knowledge [92] and is appropriate for novice programmers [91]. Next, delayed feedback is given when the student submits a solution to an auto-grader or test case.

This type of feedback is most used to show correct behavior rather than subgoals for a task [91]. Finally, feedback on demand, where the student has to ask for help explicitly, an action that novice programmers need help with [91].

Feedback can also be categorized by type. For instance, “Verification”, which informs the learners about the correctness of their responses; “Correct Response”, that tells the learner of the correct answer to a specific problem, with no additional information; “Try Again”, informs the learner about an incorrect response and allows the learner one or more attempts to answer it; “Error Flagging”, which highlights errors in a solution without giving a correct answer; “Elaborated”, explains why a specific response was correct or not and may allow the learner to review part of the instruction; “Hints”, indicate what to do next, avoiding explicitly presenting the correct answer; And “Bugs/misconceptions”, provides information about the learner’s specific errors or misconceptions [92].

Successful results in learning, student engagement, and motivation in computer programming using feedback are reported [88, 89]. For instance, Reis et al. [94] report that students using Clara, a tool that provides hints, could significantly reduce their effort to get the correct solution compared to using Python Tutor [95], which produces code visualization, or only test cases. Additionally, students scored Clara as more useful than test cases to fix bugs in their programs [96]. Also, Marwan et al. [91] presented an adaptive immediate feedback system integrated into a block-based programming environment. This system provides positive and corrective feedback in real-time as students work. The results show that the feedback system increased students’ intention to persist in CS and that students that used the feedback system had greater engagement than the students that did not use the feedback system.

In terms of expert-authored feedback, Gerdes et al. [97], presents *Ask-Elle*, a tutor for learning that supports the stepwise development of Haskell programs by providing

hints during the development process. Also using expert-authored feedback, Benotti et al. [98], presents *Mumuki*, a web-based tool that provides formative feedback. One of the differences between *Mumuki* and *Ask-Elle* is that *Mumuki* shows feedback only when the solution is submitted and not during the development of the program.

8.2 Background


8.2.1 Domain-Specific Immediate Feedback


In a previous iteration of *Code Beats*, reported on Chapter 7, I noticed that even using scaffolded, short tracks, the code developed by many students, while accurate from a computer science perspective, often violated musical guidelines, playing notes out-of-key or in an odd rhythm. While most students could hear, and correct, the most egregious musical mistakes, they often struggled with more subtle issues. This eroded their confidence overall, even when they were mastering the computer programming concepts. With that in mind, I implemented a domain-specific, immediate feedback system to guide students as they completed the activities. The feedback messages point out the music requirements of the activity, specifically where the current composition falls short, and not the coding requirements. In the context of this work, aligned with the feedback types explained by Shute [92], formative feedback will be called “Hints”, indicating what to do next, avoiding presenting the correct answer. The other type will be called “Feedback”, combining “Verification” and “Error Flagging” from their original types, indicating whether the program is correct or incorrect.

The content of the hints and feedback messages is a breakdown of the activity requirements. When the activity starts, the messages on the screen are hints that inform students of the activity requirements. At the bottom of Figure 27, in yellow,

are examples of hints, each pointing to one of the activity's requirements. The first reminds the student that the melody must be four beats long, the second states that the melody must be the same as the original, and the third states that the rhythm must be the same as the original melody. The sound of the original track (e.g., melody or drum track) is provided to students as an example.

```
1  from code_cell import *
2  """
3  FIRST TRACK - MELODY
4  Complete the melody using the playNote function and MIDI numbers 61, 68 and 69.
5  Your melody should have 4 beats. (Each playNote command is 1 beat.)
6  Your melody must match the original beat.
7  """
8
9  playNote(61)
10
```

 Make a melody with 4 beats!

 Match the original melody!


 Match the original rhythm!

Fig. 27.: Example of Hints

Each activity has a trigger that transforms the hint into feedback. In our running example, the feedback mechanism is triggered each time the number of beats in a track changes (i.e., the student plays a note or adds a rest). For example, Figure 28 shows an example of the same activity that is now four beats long. The messages that were hints in Figure 27 (yellow) now are feedback in Figure 28 (green and red). The first and third messages, in green, are examples of verification feedback indicating that the student has achieved two requirements: (1) the melody is four beats long; (2) the

rhythm is the same as the rhythm from the original melody. The second message, in red, is an example of verification feedback indicating that the student did not reach that requirement because the melody is not the same as the melody from the original song. The feedback points out that something is wrong and indicates the line of code with the problem, also working as error flagging.

```
1  from code_cell import *
2  """
3  FIRST TRACK - MELODY
4  Complete the melody using the playNote function and MIDI numbers 61, 68 and 69.
5  Your melody should have 4 beats. (Each playNote command is 1 beat.)
6  Your melody must match the original beat.
7  """
8
9  playNote(61)
10 playNote(68)
11 playNote(68)
12 playNote(69)
```

✓

 Make a melody with 4 beats!

✗

 Match the original melody! - Check line(s): [11, 12]

✓

 Match the original rhythm!

Fig. 28.: Example of Feedback - 1

On the other hand, if the student reaches all the requirements, all messages will be green, using the verification feedback to indicate that the solution is correct. Figure 29 shows an example of the same activity with all feedback messages in green, indicating that the student reached the correct answer.

Observing the classification described by Narciss [99] and extended by Keuning, Jeuring, and Heeren [88], the formative feedback used in the context of this work have

```
1  from code_cell import *
2  """
3  FIRST TRACK - MELODY
4  Complete the melody using the playNote function and MIDI numbers 61, 68 and 69.
5  Your melody should have 4 beats. (Each playNote command is 1 beat.)
6  Your melody must match the original beat.
7  """
8
9  playNote(61)
10 playNote(68)
11 playNote(69)
12 playNote(68)
```

✓

 Make a melody with 4 beats!

✓

 Match the original melody!

✓

 Match the original rhythm!

Fig. 29.: Example of Feedback - 2

the following components: Knowledge of Performance (KP), as we identify the activity subgoals and indicate when they are achieved; Knowledge of Result/Response (KR), identifying the specific subgoal as correct or incorrect; Knowledge of the Correct Results (KCR), identifying when all subgoals are correct, the activity is correct; Knowledge About Task Constraints (KTC), subtype Hints on Task Requirements (TR), breaking down the activity goal in domain-specific hints. Knowledge About Mistakes (KM), subtype Solution Errors (SE), indicating that the solution does not show the behavior expected in the activity, pointing to the line where the error is. Finally, in terms of technique, also defined by Keuning, Jeuring, and Heeren [88], the feedback system reported here uses Basic Static Analysis (BSA), analyzing the piece of code the student is writing in a specific cell to generate the hint or feedback

message.

To implement this immediate feedback, a test case-like code performs the static analysis using as input the cell code and cell output (e.g., MIDI numbers), returning to the activity cell the messages content and its type. This analysis is performed at every line completion in the students' code (e.g., hitting enter, changing line, or "playing" the cell's code). This categorizes the feedback as immediate, as it is there since the beginning of the activities, and it is updated at every code change. The feedback is expert-authored, where the rules are specified according to the expected results of each activity known by the feedback author.

8.3 Research Questions

RQ1: How does domain-specific, immediate feedback affect the activities' correctness during class?

RQ2: What is the student's perception of the domain-specific, immediate feedback?

8.4 Methods

8.4.1 Study Design and Data

This chapter reports a quasi-experiment using data collected from a *Code Beats* summer camp held in the Summer of 2022. The classes were in person and held for five days, with two classes each day, each with a duration of one hour and forty-five minutes. Each class consisted of a mix of instructions and coding activities. The coding activities are actual hip-hop songs transcribed into TunePad. The project presented to the student is almost complete, with all song tracks but one that the students will create (e.g., melody or hi-hat track). The activities were divided into two types. The first is the short activity, which asks students to create a song track

in a specific code cell that will mimic the original song. This short activity has only one correct answer, for example, a sequence of musical notes in a particular order and rhythm. And the second is the long activity that asks students to create a song track in a code cell that will fit the original song. This activity has the music requirements but allows multiple solutions. For instance, they require that the students use a group of musical notes but do not specify the order. Each class, from class one to class eight, had two short and one long activity, summing up to 24 activities ¹. The last two classes did not have these activities, as the classes were used to prepare the students to create their final project.

During the Summer of 2022, two sessions of *Code Beats* were offered, one in the morning and another in the afternoon. The student chose the session that they would like to attend. Students from one session received the activities with the immediate feedback system, and students from the other session received the same activities, except by the immediate feedback system. All the rest of the classes were the same. The group of students that received the immediate feedback will be called “Group A” and the group of students that did not receive the immediate feedback will be called “Group B”.

To analyze the correctness of the students’ solution (RQ1), the final solution developed by the student and code snapshots generated during the solving process were collected. To analyze the immediate feedback system effect (RQ2), the students from Group A answered the question: “When you saw a message that looked like this, how did it affect your motivation?” presented with each of the hints/feedback examples. Additionally, they participated in a focus group to talk about their experience.

¹The list of activities can be accessed here: <http://bit.ly/3GBp0P1>

8.4.2 Participant Demographics

Students from both groups answered demographic questions. 24 students from each group answered questions. On average, Group A students are 12.1 years old (min - 10; max - 15). 33.3% of the students from Group A self-declared as girls, 58.3% as boys, and 8.3% prefer not to say. 12.5% of the students from Group A self-declared as Asian/Asian-Americans, 33.3% as Black/African-American, 8.3% as Multi-Racial, 37.5% as White/Caucasian, and 8.3% prefer not to say. 58.3% of the students from Group A declared that they play instruments, and for 29.2% of the students from Group A, *Code Beats* was their first coding experience.

On average, Group B students are 12.2 years old (min - 10; max - 17). 16.7% of the students from Group B self-declared as girls, 75.0% as boys, and 8.3% prefer not to say. 25.0% of the students from Group B self-declared as Asian/Asian-Americans, 25.0% as Black/African-American, 8.3% as Multi-Racial, 33.3% as White/Caucasian, and 8.3% prefer not to say. 54.2% of the students from Group B declared that they play instruments, and for 16.7% of the students from Group B, *Code Beats* was their first coding experience.

8.4.3 Data Analysis

To answer the first research question, all activities' final solutions and code snapshots collected during the solving process were analyzed, searching for the code that meets the expected solution. If the code that meets the expected solution was identified at any time, the solution for that student and activity was classified as "Correct", even if the student changed the solution afterward. Otherwise, the student's solution for that activity was classified as "Incorrect". This code analysis was performed automatically using a Python script that compared the result produced by all the

solutions with the expected result.

That analysis was performed for solutions developed by students from both groups. To test the statistical significance of the difference between both groups, the Two Sample t-test was performed when the data is normally distributed (Shapiro-Wilk test with $p\text{-value} > 0.05$), and the Wilcoxon Rank Sum test when the data is non-normally distributed (Shapiro-Wilk test with $p\text{-value} < 0.05$).

To answer the second question, I analyzed the data from a post-camp survey question that asks about students' motivation regarding the immediate feedback system and a follow-up question on the post-camp interview. These questions were asked only for students from Group A.

8.5 Results

8.5.1 RQ1: Correctness

To answer the first research question, I present the results of the analysis of the solutions, where the students' solutions for each activity were classified as "Correct" or "Incorrect". There were a total of 24 activities for each group. A total of 526 solutions were submitted by students from Group A, with an average of 21.9 (SD = 1.2). A total of 565 solutions were submitted by students from Group B, with an average of 23.5 (SD = 2.4).

Figure 30 shows the percentage of correct solutions for both groups. The percentage of students that reached a correct solution is normally distributed for both groups, Group A ($p\text{-value} = 0.2117$) and Group B ($p\text{-value} = 0.1645$). The difference in the percentage of correct solutions between groups A and B is statistically significant (Two Sample t-test - $p\text{-value} = 0.0002145$).

Considering only students from Group A, on average, 59.4% of the students

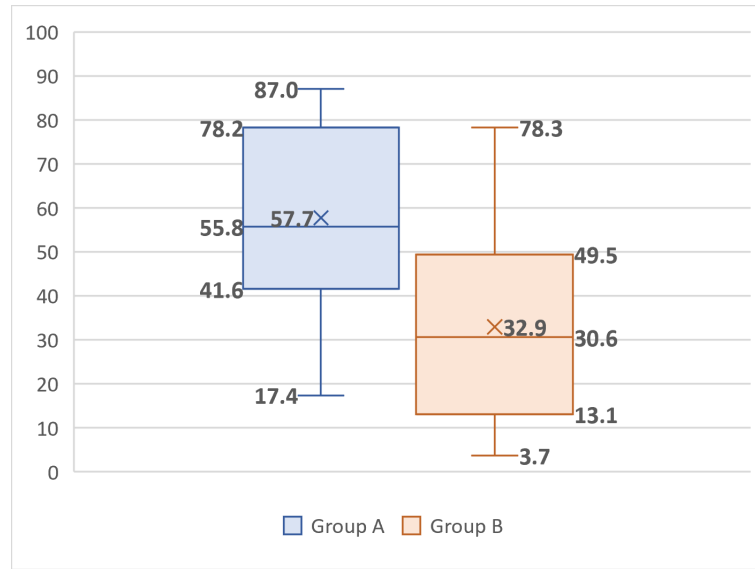


Fig. 30.: Percentage of Correct Solutions

that reached the correct solution were able to do that without receiving any error flagging during the solving process of the activity that they got right. (SD = 21.82; Shapiro-Wilk test p-value = 0.7625).

Analyzing only the long activities, the difference is more expressive. The average percentage of students that got the correct solution is 45.7% for Group A (SD = 15.7) and 13.5% for Group B (SD = 9.4). This difference is statistically significant (Wilcoxon Rank Sum test - p-value = 0.002742).

8.5.2 RQ2: Students Perception on Feedback

To answer the second research question, I present in Figure 31 the results of students' responses regarding their motivation for each message type. With examples of each message type, the student answered if they felt less motivated, more motivated, or the motivation did not change when seeing a message like the one showed them as an example.

After the camp, Group A students were asked, "how did you know if you coded

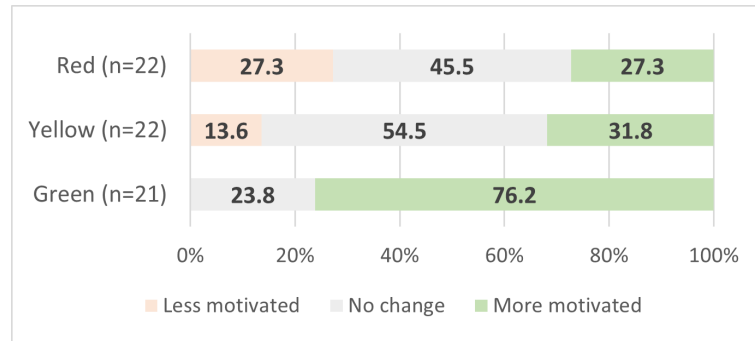


Fig. 31.: Students Motivation - Feedback System

something correctly or not.” One student related the whole feedback system behavior, since the error identification to the indication when the problem was fixed: *“Well, there’s something that pops up as an error. It’s at the bottom, of yourself, and then you have to read what the problem is or it’ll tell you the problem. Then when you find the problem, you could fix it and then it would just go away. I guess that would be how it works.”*

Another student pointed out the error flagging functionality that tells the student where the problem is: *“It would tell me the line that was in or row. It would be like, “Error in line 36,” or something. Then I’d go to 36 and see what the problem is, or there would be where it would say an error, but it isn’t actually an error. You just have to continue the code.”*

Finally, one student reported the verification behavior of the feedback system, making a correlation with a check mark: *“It basically gives you this thing on the bottom part where it gives you like a check mark, if it’s correct. And if it has anything bad on it, then it says invalid syntax or something.”*

Furthermore, during the focus group, they were prompted to discuss the feedback they received, how helpful they were in identifying and fixing mistakes, and what would happen if TunePad had no feedback available.

One student pointed out how the feedback helped him/her to identify the MIDI numbers that were expected: *“The hints were helpful when you were trying to figure out, when you were trying to... when you knew what we wanted it to sound like, but you had to find out which number on the keyboard or which string on the guitar it was. And the hints’ kind of helped you lean towards the area that the number you wanted were.”*

Some students highlighted the importance of the feedback to help develop a good-sounding solution, for example: *“If they weren’t there, then the music would either sound very bad or not play at all.”*, and: *“Then until then when you finished the music, you’re wondering, “How come it’s not working,” or it sounds so off and you would have never found it because the error wouldn’t have showed you.”*

Another student mentioned how hard it would be to find the lines with error if the feedback system was not in use: *“It would’ve been a lot harder [without the hints]... Because you don’t know what you did wrong, so you would have to check every single line of code you have to see what you did wrong. Because it tells you where exactly you did it wrong.”*

Finally, one student stated that it would not be possible to know what happened with its solution: *“You would never know what happened.”*, and another student thinking of a bigger solution stated that with more lines of code the situation would be worst: *“If you had larger lines that go up to 100, then we would be doomed.”*

8.6 Discussion and Conclusion

The results presented in this chapter provide evidence that using domain-specific immediate feedback improves the percentage of correct solutions developed by the students. The data show a statistically significant difference in the percentage of students that reached the expected solution between both groups. Students who received

domain-specific immediate feedback have a higher percentage of correct solutions than those who did not receive the feedback.



The percentage of students who reached the correct solution is greater for the group that received the immediate feedback than for the group that did not receive feedback.

This finding differs in parts from the one reported by Reis et al. [96], where students using the feedback system and only test cases got similar scores in a post-test. It is crucial to mention that the measure used in that work differs from the measure I am using, as I am using specifically the activity correctness and not a post-test. Conversely, Marwan et al. [91], found suggestive evidence that using their feedback system improved students' performance and learning. The finding is reinforced by Marwan, Williams, and Price [100], which found evidence suggesting that code hints with textual explanations and code hints with both textual explanations and self-explanations prompts significantly improve performance.

Furthermore, when isolating and analyzing only the long activities, this difference between the averages of the percentage of students that reached the correct solutions from both groups is even higher, indicating that the feedback is even more critical with longer tasks, showing that students are more engaged in doing the activities. The difference in student engagement aligns with findings from Marwan et al. [91], where students using their feedback system significantly improved their engagement. However, it is essential to mention that the measure used in that work differs from the measure used here.

The results presented in this chapter show that students were more motivated by the “green” messages that indicated that they met one requirement. In contrast, the motivation did not change when students saw a “yellow” or “red” message. This

suggests that, regardless of the hints and feedback indicating something was wrong, the students were motivated to see that their solution was correct.



Most students were motivated to receive the “green” feedback during the activity solution process.

This finding is in some measure related to the result reported by Mitrovic, Ohlsson, and Barrow [101], not associated with the motivation but with the importance of the positive feedback, as they report the impact of the positive feedback on students’ performance on SQL activities. Moreover, this finding is related to the affective consequences of feedback, where positive feedback increases motivation, carrying information about one’s accomplishments, strengths, and correct responses [102].

Analyzing the interviews, it is possible to conclude that students understood how immediate feedback works and how to use it to reach the objectives of the activities. Furthermore, students believe the feedback is helpful, and without it, it would be much harder to solve the activities.



Students found the hints and feedback helpful, and it would be much harder to solve the problems without them.

This finding is in line with what is related by Reis et al. [96], where students score their hints system as more useful than test cases, and the students could easily find the location of the errors using the hints system.

8.7 Limitation and Threats to Validity

Due to the nature of the summer camp organization, where each student chooses to register for the session that best suits their interests, it was impossible to have an equivalent distribution of participants between both groups or randomize the stu-

dents' selection, that is the main reason to use a quasi-experimental setup rather than a controlled experiment. This immediate feedback system was implemented specifically in TunePad. Therefore, implementing a similar approach in other tools used to teach coding with music or in different contexts might not be possible.

In this study, the use of immediate feedback directs students toward a solution that meets the activity requirements. However, it is important to note that utilizing different forms of feedback may yield varying results. Additionally, it is uncertain whether the findings from this study can be replicated with different demographics.

CHAPTER 9

CONCLUSION

The primary goal of this research presented in this dissertation is to design and evaluate a curriculum that uses hip-hop beats to teach the foundational concepts of computer programming to attract and engage students in the CS field, helping the growing body of work of studies that look for alternatives that attract those who are not yet interested in the field, presenting an alternative to the traditional approaches. This chapter summarizes the major research contributions and findings from the research studies.

9.1 Research Contributions

Designed and implemented multiple instances of the *Code Beats* curriculum. This dissertation described a curriculum that teaches the foundational concepts of computer programming, incorporating them with music. Its first version consisted of 15 classes, which was changed for its second version, reducing to ten classes. One of the reasons for that is that, during the first experiences with the curriculum, concepts like conditionals were not easy to map. So I decided to remove this concept and also make some adjustments in the order each concept was presented.

The curriculum presented in this dissertation aims to introduce students to different computer programming resources, including some not usually explained during introductory computer programming classes, such as modularization and parallelism. Nonetheless, the curriculum presented here is not static. On the contrary, it can be adapted and changed according to the needs. Indeed, in the experiment reported

in Chapter 6, the curriculum was adapted, removing some of the concepts to fit the conditions of that experiment best.

Studied the impact of *Code Beats* approach on student engagement in computer science in multiple contexts. One of the main objectives of *Code Beats* is to attract and engage students toward CS.

In this dissertation, the students' engagement was reported in three chapters. First, in Chapter 4, I reported the initial results of this approach, where after a virtual summer camp with students from middle school, it was possible to say that the use of hip-hop to engage students toward CS is promising. Next, in Chapter 5, with a different audience, this time applying the approach to adult learners, I documented changes in adult learners' perception of computer programming, where, after *Code Beats*, they started to think of computer programming as something they could learn and do, rather than their previous perceptions of computer programming, that was something boring, difficult, and impossible to learn. Finally, in Chapter 6, I applied the *Code Beats* approach with middle schoolers that do not have music experience. In this experiment, it was possible to see that, according to students' interviews and researchers' observations, the use of music was a crucial factor in engaging students in this course, also being fundamental to attracting them to register and attend the classes, with some students reporting that they would not be interested in participating in the classes if it were not related to music.

Investigated the use of two student support educational techniques to improve the students' learning experience. Despite the exciting and motivating student engagement results observed when using the *Code Beats* approach, we believed that the student learning environment could be improved. The first strategy I reported in Chapter 7 is using different scaffolding strategies in the in-class activities. The experiment aimed to test these historically used learning strategies in the music

coding context. This required strategies adaption and provided new findings to the CS education community, contrasting with results from previous works using these scaffolding types in different contexts. Unfortunately, even using these scaffolding strategies, the percentage of correct solutions delivered by the students was quite lower than expected, with, in some cases, the rate of correct solutions being lower than 50%.

So, after using scaffolding strategies, I explored using expert-authored domain-specific immediate feedback to guide students toward the activity solution, helping them with the musical requirements and allowing them to focus their attention on the code part of the solution. The results using feedback were exciting, improving the correct solution's rate compared to students who did not receive this extra help.

9.2 Significant Findings

Each experiment revealed specific findings to contribute to this research dissertation. Starting with the main finding from Chapter 4, that reported the first experience with the *Code Beats* approach: **The use of music, specifically hip-hop, to teach coding improved the students' engagement towards Computer Science.**

Motivated by this first finding that was observed with middle school students, I used the *Code Beats* approach with adult learners. This experience is reported in Chapter 5, and its main finding is: **Music has proven to be a helpful tool in teaching foundational concepts to adult learners, as it can alter their perception of computer programming. Prior to attending *Code Beats*, many perceived computer programming as boring, difficult, or even impossible to learn. However, after participating in the program, their outlook changed, and they began to view computer programming as an attainable skill.**

Next, the *Code Beats* approach was tested in a different context, with students without previous music knowledge. This experiment is reported in Chapter 6 and has as its main finding: **The use of music to teach coding was a key factor in attracting and engaging the students in the classes.**

Beyond the results in terms of students' motivation and engagement, I also tested education techniques to facilitate the student learning process. The first technique that I tested using the *Code Beats* approach is using activities scaffolding techniques that are commonly used in computer programming in the context of coding music. The main finding from the experiment reported in Chapter 7 is: **When coding music, students struggled to create correct solutions even with scaffold-based support.**

Finally, the use of immediate feedback on coding activities was implemented and tested in the context of *Code Beats*. The experiment is reported in Chapter 8, and its main finding is: **The use of immediate feedback during the activities-solving process improved the percentage of activities' correctness, with students perceiving the use of feedback as helpful and being motivated when seeing that their activity was correct.**

Ultimately, it is possible to say that the thesis that *introducing students to computer programming using high-quality, culturally relevant music will improve students' perceptions of the computer science field* holds.

A total of 352 students were part of the experiments reported in this dissertation, with 245 students being part of the demographics considered in each chapter. The table 10 summarizes the number of participants for each experiment.

Experiment Details		Number of Participants	
Chapter	Format	Total	For Demographics
4 - Engagement	Virtual	45	17
5 - Adult Learners	Virtual	40	32
6 - Without Music Background	In-person	55	25
7 - Scaffold-based Activities	Virtual	149	123
8 - Music-coding Feedback	In-person	63	48
Total		352	245

Table 10.: Code Beats Camps - Number of Participants

9.3 Future Work

This dissertation contributes to advancing the CS education community and the growing body of research about music to teach computer programming. That being said, there are several directions for future work.

Scaling up, collecting more data. The work reported in this dissertation proposed the initial framework of *Code Beats*, testing this approach in different contexts, such as middle school students with previous music knowledge, middle school students that do not have music experience, and adult learners. Even with the exciting results so far, more experiments can be performed, collecting more data and improving the framework.

Tweaks to the curriculum. With the new experiments, it is possible to tweak the curriculum, adding new computer programming concepts or even finding different ways to connect the already used concepts with music.

New music genres. Additional music genres can be studied and used with the

proposed curriculum. The use of different music genres has the potential to attract other populations, contributing to the effort of broadening CS education.

More qualitative studies. The studies conducted and reported in this dissertation show that the approach attracts students to attend the classes and engages them toward CS. However, more qualitative studies are necessary to understand why it works.

Analysis by demographics. The results reported in this dissertation are overall results for the population of the studies. It would be interesting to obtain the results of each subgroup of the population, for instance, by race, by gender, and by previous experience in coding or music.

Testing the students' gained knowledge. Another area for future research is to test the impact of the *Code Beats* approach on students' knowledge of music and computer programming.

References

- [1] Polina Charters et al. “Challenging Stereotypes and Changing Attitudes: The Effect of a Brief Programming Encounter on Adults’ Attitudes toward Programming”. In: *Proceedings of the 45th ACM Technical Symposium on Computer Science Education*. SIGCSE ’14. Atlanta, Georgia, USA: Association for Computing Machinery, 2014, pp. 653–658. ISBN: 9781450326056. DOI: 10.1145/2538862.2538938. URL: <https://doi.org/10.1145/2538862.2538938>.
- [2] United States Bureau of Labor Statistics. *Occupational projections and worker characteristics*. Dec. 2020. URL: <https://www.bls.gov/emp/tables/occupational-projections-and-characteristics.htm>.
- [3] Allison Scott et al. “Broadening Participation in Computer Science: Existing Out-of-School Initiatives and a Case Study”. In: *ACM Inroads* 7.4 (Nov. 2016), pp. 84–90. ISSN: 2153-2184. DOI: 10.1145/2994153. URL: <https://doi.org/10.1145/2994153>.
- [4] College Board. *AP Program Participation and Performance Data 2020*. Dec. 2020. URL: <https://research.collegeboard.org/programs/ap/data/participation/ap-2020>.
- [5] Anthony V Robins. “Novice Programmers and Introductory Programming”. In: *The Cambridge handbook of computing education research* (2019), p. 327.
- [6] Colleen M Lewis, Niraj Shah, and Katrina Falkner. “Equity and Diversity”. In: *The Cambridge handbook of computing education research* (2019), p. 481.

- [7] Jane Margolis and Allan Fisher. *Unlocking the clubhouse: Women in computing*. MIT press, 2002.
- [8] E. H. Erikson. *The life cycle completed: A review*. New York: Norton, 1982.
- [9] Eccles J. S. Simpkins S. D. Davis-Kean P. E. “Math and science motivation: A longitudinal examination of the links between choices and beliefs”. In: *Developmental Psychology* 42 (2006), p. 70.
- [10] Geneva Gay. “Teaching to and through cultural diversity”. In: *Curriculum inquiry* 43.1 (2013), pp. 48–70.
- [11] Gloria Ladson-Billings. “Toward a theory of culturally relevant pedagogy”. In: *American educational research journal* 32.3 (1995), pp. 465–491.
- [12] Bill Manaris, Blake Stevens, and Andrew R Brown. “JythonMusic: An environment for teaching algorithmic music composition, dynamic coding and musical performativity”. In: *Journal of Music, Technology & Education* 9.1 (2016), pp. 33–56.
- [13] Jeff Chang and S Craig Watkins. “It’s a hip-hop world”. In: *Foreign Policy* 163 (2007), p. 58.
- [14] United States Census Bureau. *Racial and Ethnic Diversity in the United States: 2010 Census and 2020 Census*. Aug. 2020. URL: <https://www.census.gov/library/visualizations/interactive/racial-and-ethnic-diversity-in-the-united-states-2010-and-2020-census.html>.
- [15] United States Bureau of Labor Statistics. *Labor force characteristics by race and ethnicity, 2019*. Dec. 2020. URL: <https://www.bls.gov/opub/reports/race-and-ethnicity/2019/home.htm>.

- [16] Code.org and CSTA. “2020 State of Computer Science Education - Illuminating Disparities”. In: *code.org* (2020). URL: https://advocacy.code.org/2020_state_of_cs.pdf.
- [17] Tim Bell and Jan Vahrenhold. “CS Unplugged—How Is It Used, and Does It Work?” In: *Adventures Between Lower Bounds and Higher Altitudes: Essays Dedicated to Juraj Hromkovič on the Occasion of His 60th Birthday*. Ed. by Hans-Joachim Böckenhauer, Dennis Komm, and Walter Unger. Cham: Springer International Publishing, 2018, pp. 497–521. ISBN: 978-3-319-98355-4. DOI: 10.1007/978-3-319-98355-4_29. URL: https://doi.org/10.1007/978-3-319-98355-4_29.
- [18] Jennifer S. Kay and Janet G. Moss. “Using robots to teach programming to K-12 teachers”. In: *2012 Frontiers in Education Conference Proceedings*. 2012, pp. 1–6. DOI: 10.1109/FIE.2012.6462375.
- [19] Mitchel Resnick et al. “Scratch: Programming for All”. In: *Commun. ACM* 52.11 (Nov. 2009), pp. 60–67. ISSN: 0001-0782. DOI: 10.1145/1592761.1592779. URL: <https://doi.org/10.1145/1592761.1592779>.
- [20] Brian Harvey. “Bringing ”No Ceiling” to Scratch: Can One Language Serve Kids and Computer Scientists?” In: 2010.
- [21] Bernat Romagosa i Carrasquer. “The Snap! Programming System”. In: *Encyclopedia of Education and Information Technologies*. Ed. by Arthur Tattall. Cham: Springer International Publishing, 2019, pp. 1–10. ISBN: 978-3-319-60013-0. DOI: 10.1007/978-3-319-60013-0_28-2. URL: https://doi.org/10.1007/978-3-319-60013-0_28-2.
- [22] Alexander Repenning et al. “AgentCubes: Enabling 3D Creativity by Addressing Cognitive and Affective Programming Challenges”. In: *Proceedings*

- of *EdMedia + Innovate Learning 2012*. Ed. by Tel Amiel and Brent Wilson. Denver, Colorado, USA: Association for the Advancement of Computing in Education (AACE), June 2012, pp. 2762–2771. URL: <https://www.learntechlib.org/p/41159>.
- [23] Colleen M. Lewis, Niral Shah, and Katrina Falkner. “Equity and Diversity”. In: pp. 481–510. DOI: 10.1017/9781108654555.017.
 - [24] Colleen M. Lewis, Ruth E. Anderson, and Ken Yasuhara. ““I Don’t Code All Day”: Fitting in Computer Science When the Stereotypes Don’t Fit”. In: *Proceedings of the 2016 ACM Conference on International Computing Education Research*. ICER ’16. Melbourne, VIC, Australia: Association for Computing Machinery, 2016, pp. 23–32. ISBN: 9781450344494. DOI: 10.1145/2960310.2960332. URL: <https://doi.org/10.1145/2960310.2960332>.
 - [25] Anthony V. Robins. “Novice Programmers and Introductory Programming”. In: pp. 327–376. DOI: 10.1017/9781108654555.013.
 - [26] Brian Magerko et al. “Tackling Engagement in Computing with Computational Music Remixing”. In: *Proceeding of the 44th ACM Technical Symposium on Computer Science Education*. SIGCSE ’13. Denver, Colorado, USA: Association for Computing Machinery, 2013, pp. 657–662. ISBN: 9781450318686. DOI: 10.1145/2445196.2445390. URL: <https://doi.org/10.1145/2445196.2445390>.
 - [27] Samuel Aaron and Alan F. Blackwell. “From Sonic Pi to Overtone: Creative Musical Experiences with Domain-Specific and Functional Languages”. In: *Proceedings of the First ACM SIGPLAN Workshop on Functional Art, Music, Modeling & Design*. FARM ’13. Boston, Massachusetts, USA: Association for

- Computing Machinery, 2013, pp. 35–46. ISBN: 9781450323864. DOI: 10.1145/2505341.2505346. URL: <https://doi.org/10.1145/2505341.2505346>.
- [28] Jamie Gorson et al. “TunePad: Computational Thinking Through Sound Composition”. In: *Proceedings of the 2017 Conference on Interaction Design and Children*. ACM. 2017, pp. 484–489.
- [29] Jason Freeman et al. “Engaging Underrepresented Groups in High School Introductory Computing through Computational Remixing with EarSketch”. In: *Proceedings of the 45th ACM Technical Symposium on Computer Science Education*. SIGCSE ’14. Atlanta, Georgia, USA: Association for Computing Machinery, 2014, pp. 85–90. ISBN: 9781450326056. DOI: 10.1145/2538862.2538906. URL: <https://doi.org/10.1145/2538862.2538906>.
- [30] Brian Magerko et al. “EarSketch: A STEAM-Based Approach for Underrepresented Populations in High School Computer Science Education”. In: *ACM Trans. Comput. Educ.* 16.4 (Sept. 2016). DOI: 10.1145/2886418. URL: <https://doi.org/10.1145/2886418>.
- [31] Christian Köppe. “Program a Hit – Using Music as Motivator for Introducing Programming Concepts”. In: *Proceedings of the 2020 ACM Conference on Innovation and Technology in Computer Science Education*. ITiCSE ’20. Trondheim, Norway: Association for Computing Machinery, 2020, pp. 266–272. ISBN: 9781450368742. DOI: 10.1145/3341525.3387377. URL: <https://doi.org/10.1145/3341525.3387377>.
- [32] Christopher Petrie. “Programming music with Sonic Pi promotes positive attitudes for beginners”. In: *Computers & Education* 179 (2022), p. 104409. ISSN: 0360-1315. DOI: <https://doi.org/10.1016/j.compedu.2021.104409>.

104409. URL: <https://www.sciencedirect.com/science/article/pii/S0360131521002864>.
- [33] Michael Horn et al. “TunePad: Engaging learners at the intersection of music and code”. In: (2020).
- [34] Tom McKlin et al. “Leveraging Prior Computing and Music Experience for Situational Interest Formation”. In: *Proceedings of the 52nd ACM Technical Symposium on Computer Science Education*. SIGCSE ’21. Virtual Event, USA: Association for Computing Machinery, 2021, pp. 928–933. ISBN: 9781450380621. DOI: 10.1145/3408877.3432431. URL: <https://doi.org/10.1145/3408877.3432431>.
- [35] Allison Scott et al. “Toward culturally responsive computing education”. In: *ACM Inroads* 7.4 (Nov. 2013), pp. 84–90. ISSN: 2153-2184. DOI: 10.1145/2994153. URL: <https://doi.org/10.1145/2994153>.
- [36] Jessica Morales-Chicas et al. “Computing with Relevance and Purpose: A Review of Culturally Relevant Education in Computing”. In: *International Journal of Multicultural Education* 21.1 (Mar. 2019), pp. 125–155. DOI: 10.18251/ijme.v21i1.1745. URL: <https://ijme-journal.org/index.php/ijme/article/view/1745>.
- [37] Code.org. *Code.org Computer Science Principles Syllabus and Overview*. 2020. URL: <https://code.org/files/CSPSyllabus2020.pdf>.
- [38] Samuel Aaron, Alan F. Blackwell, and Pamela Burnard. “The development of Sonic Pi and its use in educational partnerships: Co-creating pedagogies for learning computer programming”. In: *Journal of Music, Technology and Education* 9.1 (2016), pp. 75–94. ISSN: 1752-7066. DOI: doi:10.1386/jmte.

- 9.1.75_1. URL: <https://www.ingentaconnect.com/content/intellect/jmte/2016/00000009/00000001/art00006>.
- [39] Mike Horn, Amartya Banerjee, and Matthew Brucker. “TunePad Playbooks: Designing Computational Notebooks for Creative Music Coding”. In: *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems*. CHI ’22. New Orleans, LA, USA: Association for Computing Machinery, 2022. ISBN: 9781450391573. DOI: 10.1145/3491102.3502021. URL: <https://doi.org/10.1145/3491102.3502021>.
 - [40] Irene Lee et al. “Computational Thinking for Youth in Practice”. In: *ACM Inroads* 2.1 (Feb. 2011), pp. 32–37. ISSN: 2153-2184. DOI: 10.1145/1929887.1929902. URL: <https://doi.org/10.1145/1929887.1929902>.
 - [41] G.W. Corder and D.I. Foreman. *Nonparametric Statistics for Non-Statisticians: A Step-by-Step Approach*. Wiley, 2009. ISBN: 9780470454619. URL: <https://books.google.com/books?id=-uf0fzVp6qYC>.
 - [42] Philip J. Guo. “Older Adults Learning Computer Programming: Motivations, Frustrations, and Design Opportunities”. In: *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*. CHI ’17. Denver, Colorado, USA: Association for Computing Machinery, 2017, pp. 7070–7083. ISBN: 9781450346559. DOI: 10.1145/3025453.3025945. URL: <https://doi.org/10.1145/3025453.3025945>.
 - [43] Seymour A Papert. *Mindstorms: Children, computers, and powerful ideas*. Basic books, 2020.
 - [44] Quinn Burke et al. “Understanding the Software Development Industry’s Perspective on Coding Boot Camps versus Traditional 4-Year Colleges”. In: *Proceedings of the 49th ACM Technical Symposium on Computer Science Edu-*

- cation. SIGCSE '18. Baltimore, Maryland, USA: Association for Computing Machinery, 2018, pp. 503–508. ISBN: 9781450351034. DOI: 10.1145/3159450.3159485. URL: <https://doi.org/10.1145/3159450.3159485>.
- [45] Sherry Seibel and Nanette Veilleux. “Factors influencing women entering the software development field through coding bootcamps vs. computer science bachelor’s degrees”. In: *Journal of Computing Sciences in Colleges* 34.6 (2019), pp. 84–96.
- [46] Kyle Thayer and Amy J. Ko. “Barriers Faced by Coding Bootcamp Students”. In: *Proceedings of the 2017 ACM Conference on International Computing Education Research*. ICER '17. Tacoma, Washington, USA: Association for Computing Machinery, 2017, pp. 245–253. ISBN: 9781450349680. DOI: 10.1145/3105726.3106176. URL: <https://doi.org/10.1145/3105726.3106176>.
- [47] Maren Krafft, Gordon Fraser, and Neil Walkinshaw. “Motivating Adult Learners by Introducing Programming Concepts with Scratch”. In: *Proceedings of the 4th European Conference on Software Engineering Education*. ECSEE '20. Seon/Bavaria, Germany: Association for Computing Machinery, 2020, pp. 22–26. ISBN: 9781450377522. DOI: 10.1145/3396802.3396818. URL: <https://doi.org/10.1145/3396802.3396818>.
- [48] Sergio Sayago and Ángel Bergantiños. “Exploring the first experiences of computer programming of older people with low levels of formal education: A participant observational case study”. In: *International Journal of Human-Computer Studies* 148 (2021), p. 102577. ISSN: 1071-5819. DOI: <https://doi.org/10.1016/j.ijhcs.2020.102577>. URL: <https://www.sciencedirect.com/science/article/pii/S1071581920301798>.

- [49] Virginia Braun and Victoria Clarke. “Using thematic analysis in psychology”. In: *Qualitative Research in Psychology* 3.2 (2006), pp. 77–101. DOI: 10.1191/1478088706qp063oa. URL: <https://www.tandfonline.com/doi/abs/10.1191/1478088706qp063oa>.
- [50] Douglas Lusa Krug et al. “Code Beats: A Virtual Camp for Middle Schoolers Coding Hip Hop”. In: *Proceedings of the 52nd ACM Technical Symposium on Computer Science Education*. New York, NY, USA: Association for Computing Machinery, 2021, pp. 397–403. ISBN: 9781450380621. URL: <https://doi.org/10.1145/3408877.3432424>.
- [51] Merijke Coenraad, Bih Janet Fofang, and David Weintrop. “Gusanos y Esferos: Computing with Youth in Rural El Salvador”. In: *Proceedings of the 52nd ACM Technical Symposium on Computer Science Education*. SIGCSE ’21. Virtual Event, USA: Association for Computing Machinery, 2021, pp. 404–410. ISBN: 9781450380621. DOI: 10.1145/3408877.3432535. URL: <https://doi.org/10.1145/3408877.3432535>.
- [52] James Zhang et al. “Scientific Collaboration Network Analysis for Computing Education Conferences”. In: *Proceedings of the 26th ACM Conference on Innovation and Technology in Computer Science Education V. 1*. ITiCSE ’21. Virtual Event, Germany: Association for Computing Machinery, 2021, pp. 582–588. ISBN: 9781450382144. DOI: 10.1145/3430665.3456385. URL: <https://doi.org/10.1145/3430665.3456385>.
- [53] Mikko Apiola et al. “Computing Education Research Compiled: Keyword Trends, Building Blocks, Creators, and Dissemination”. In: *IEEE Access* 10 (2022), pp. 27041–27068. DOI: 10.1109/ACCESS.2022.3157609.

- [54] Francisco J. Gutierrez et al. “Coding or Hacking? Exploring Inaccurate Views on Computing and Computer Scientists among K-6 Learners in Chile”. In: *Proceedings of the 49th ACM Technical Symposium on Computer Science Education*. SIGCSE '18. Baltimore, Maryland, USA: Association for Computing Machinery, 2018, pp. 993–998. ISBN: 9781450351034. DOI: 10.1145/3159450.3159598. URL: <https://doi.org/10.1145/3159450.3159598>.
- [55] André Branco et al. “Programming for Children and Teenagers in Brazil: A 5-Year Experience of an Outreach Project”. In: *Proceedings of the 52nd ACM Technical Symposium on Computer Science Education*. SIGCSE '21. Virtual Event, USA: Association for Computing Machinery, 2021, pp. 411–417. ISBN: 9781450380621. DOI: 10.1145/3408877.3432554. URL: <https://doi.org/10.1145/3408877.3432554>.
- [56] Yifan Zhang et al. “A Case Study of Middle Schoolers’ Use of Computational Thinking Concepts and Practices during Coded Music Composition”. In: *Proceedings of the 27th ACM Conference on on Innovation and Technology in Computer Science Education Vol. 1*. ITiCSE '22. Dublin, Ireland: Association for Computing Machinery, 2022, pp. 33–39. ISBN: 9781450392013. DOI: 10.1145/3502718.3524757. URL: <https://doi.org/10.1145/3502718.3524757>.
- [57] Brasil and Ministério da Educação. *Catálogo Nacional de Cursos Técnicos*. 2016.
- [58] Leila Ribeiro et al. “Diretrizes da Sociedade Brasileira de Computação para o Ensino de Computação na Educação Básica”. In: *Sociedade Brasileira de Computação* (2019).

- [59] Taciana Pontual Falcão. “Computational Thinking for All: What Does It Mean for Teacher Education in Brazil?” In: *Anais do Simpósio Brasileiro de Educação em Computação*. SBC. 2021, pp. 371–379.
- [60] Priscila S. C. Santos, Luis Gustavo J. Araujo, and Roberto A. Bittencourt. “A Mapping Study of Computational Thinking and Programming in Brazilian K-12 Education”. In: *2018 IEEE Frontiers in Education Conference (FIE)*. 2018, pp. 1–8. DOI: 10.1109/FIE.2018.8658828.
- [61] Cristiano Maciel, Sílvia Amélia Bim, and Karen da Silva Figueiredo. “Digital Girls Program: Disseminating Computer Science to Girls in Brazil”. In: *Proceedings of the 1st International Workshop on Gender Equality in Software Engineering*. GE ’18. Gothenburg, Sweden: Association for Computing Machinery, 2018, pp. 29–32. ISBN: 9781450357388. DOI: 10.1145/3195570.3195574. URL: <https://doi.org/10.1145/3195570.3195574>.
- [62] Núcleo de Informação e Coordenação do Ponto BR. *Survey on the use of information and communication technologies in Brazilian schools : ICT in Education 2020*. 2021. URL: <https://cetic.br/pt/publicacao/pesquisa-sobre-o-uso-das-tecnologias-de-informacao-e-comunicacao-nas-escolas-brasileiras-tic-educacao-2020/>.
- [63] Gabriel SantClair, Julia Godinho, and Janaína Gomide. “Affordable Robotics Projects in Primary Schools: A Course Experience in Brazil”. In: *Proceedings of the 52nd ACM Technical Symposium on Computer Science Education*. SIGCSE ’21. Virtual Event, USA: Association for Computing Machinery, 2021, pp. 66–72. ISBN: 9781450380621. DOI: 10.1145/3408877.3432555. URL: <https://doi.org/10.1145/3408877.3432555>.

- [64] Liane Hentschke. “Global Policies and Local Needs of Music Education in Brazil”. In: *Arts Education Policy Review* 114.3 (2013), pp. 119–125. DOI: 10.1080/10632913.2013.803415. eprint: <https://doi.org/10.1080/10632913.2013.803415>. URL: <https://doi.org/10.1080/10632913.2013.803415>.
- [65] Dwight Manning and Marilia Kamil. “New legislation in Brazilian music education: Studying the law and its implementation”. In: *International Journal of Music Education* 35.1 (2017), pp. 79–92. DOI: 10.1177/0255761415619422. eprint: <https://doi.org/10.1177/0255761415619422>. URL: <https://doi.org/10.1177/0255761415619422>.
- [66] Gustavo Cunha de Araújo and Irany Ferreira Lima. “Gaps in the training of arts teachers: old challenges and problems in Brazilian education”. In: *Arts Education Policy Review* 123.4 (2022), pp. 178–193. DOI: 10.1080/10632913.2020.1844830. eprint: <https://doi.org/10.1080/10632913.2020.1844830>. URL: <https://doi.org/10.1080/10632913.2020.1844830>.
- [67] Chrystalla Mouza et al. “Development, Implementation, and Outcomes of an Equitable Computer Science After-School Program: Findings From Middle-School Students”. In: *Journal of Research on Technology in Education* 48.2 (2016), pp. 84–104. DOI: 10.1080/15391523.2016.1146561. eprint: <https://doi.org/10.1080/15391523.2016.1146561>. URL: <https://doi.org/10.1080/15391523.2016.1146561>.
- [68] Barbara Ericson and Tom McKlin. “Effective and sustainable computing summer camps”. In: *Proceedings of the 43rd ACM technical symposium on Computer Science Education*. 2012, pp. 289–294.

- [69] Mark Guzdial. “Software-realized scaffolding to facilitate programming for science learning”. In: *Interactive learning environments* 4.1 (1994), pp. 001–044.
- [70] Barbara J. Ericson, Lauren E. Margulieux, and Jochen Rick. “Solving Parsons Problems versus Fixing and Writing Code”. In: *Proceedings of the 17th Koli Calling International Conference on Computing Education Research*. Koli Calling ’17. Koli, Finland: Association for Computing Machinery, 2017, pp. 20–29. ISBN: 9781450353014. DOI: 10 . 1145 / 3141880 . 3141895. URL: <https://doi.org/10.1145/3141880.3141895>.
- [71] Dale Parsons and Patricia Haden. “Parson’s Programming Puzzles: A Fun and Effective Learning Tool for First Programming Courses”. In: *Proceedings of the 8th Australasian Conference on Computing Education - Volume 52*. ACE ’06. Hobart, Australia: Australian Computer Society, Inc., 2006, pp. 157–163. ISBN: 1920682341.
- [72] Jean M. Griffin. “Learning by Taking Apart: Deconstructing Code by Reading, Tracing, and Debugging”. In: *Proceedings of the 17th Annual Conference on Information Technology Education*. SIGITE ’16. Boston, Massachusetts, USA: Association for Computing Machinery, 2016, pp. 148–153. ISBN: 9781450344524. DOI: 10 . 1145 / 2978192 . 2978231. URL: <https://doi.org/10.1145/2978192.2978231>.
- [73] Kyle J. Harms, Noah Rowlett, and Caitlin Kelleher. “Enabling independent learning of programming concepts through programming completion puzzles”. In: *2015 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. 2015, pp. 271–279. DOI: 10.1109/VLHCC.2015.7357226.

- [74] Brian J Reiser. “Scaffolding complex learning: The mechanisms of structuring and problematizing student work”. In: *The Journal of the Learning sciences* 13.3 (2004), pp. 273–304.
- [75] Heidi Webb and Mary Beth Rosson. “Using Scaffolded Examples to Teach Computational Thinking Concepts”. In: *Proceeding of the 44th ACM Technical Symposium on Computer Science Education*. SIGCSE ’13. Denver, Colorado, USA: Association for Computing Machinery, 2013, pp. 95–100. ISBN: 9781450318686. DOI: 10.1145/2445196.2445227. URL: <https://doi.org/10.1145/2445196.2445227>.
- [76] Sue Sentance, Jane Waite, and Maria Kallia. “Teachers’ Experiences of Using PRIMM to Teach Programming in School”. In: SIGCSE ’19. Minneapolis, MN, USA: Association for Computing Machinery, 2019, pp. 476–482. ISBN: 9781450358903. DOI: 10.1145/3287324.3287477. URL: <https://doi.org/10.1145/3287324.3287477>.
- [77] Veronica Cateté, Amy Isvik, and Tiffany Barnes. “Infusing Computing: A Scaffolding and Teacher Accessibility Analysis of Computing Lessons Designed by Novices”. In: *Koli Calling ’20: Proceedings of the 20th Koli Calling International Conference on Computing Education Research*. Koli Calling ’20. Koli, Finland: Association for Computing Machinery, 2020. ISBN: 9781450389211. DOI: 10.1145/3428029.3428056. URL: <https://doi.org/10.1145/3428029.3428056>.
- [78] Nicholas Lytle et al. “Position: Scaffolded Coding Activities Afforded by Block-Based Environments”. In: *2019 IEEE Blocks and Beyond Workshop (B B)*. 2019, pp. 5–7. DOI: 10.1109/BB48857.2019.8941203.

- [79] Petri Ihantola and Ville Karavirta. “Two-Dimensional Parson’s Puzzles: The Concept, Tools, and First Observations”. In: *Journal of Information Technology Education: Innovations in Practice* 10 (Jan. 2011), pp. 1–14. DOI: 10.28945/1394.
- [80] Barbara J. Ericson, Mark J. Guzdial, and Briana B. Morrison. “Analysis of Interactive Features Designed to Enhance Learning in an Ebook”. In: *Proceedings of the Eleventh Annual International Conference on International Computing Education Research*. ICER ’15. Omaha, Nebraska, USA: Association for Computing Machinery, 2015, pp. 169–178. ISBN: 9781450336307. DOI: 10.1145/2787622.2787731. URL: <https://doi.org/10.1145/2787622.2787731>.
- [81] Rui Zhi et al. “Evaluating the Effectiveness of Parsons Problems for Block-Based Programming”. In: *Proceedings of the 2019 ACM Conference on International Computing Education Research*. ICER ’19. Toronto ON, Canada: Association for Computing Machinery, 2019, pp. 51–59. ISBN: 9781450361859. DOI: 10.1145/3291279.3339419. URL: <https://doi.org/10.1145/3291279.3339419>.
- [82] Michael J. Lee. “Gidget: An online debugging game for learning and engagement in computing education”. In: *2014 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. 2014, pp. 193–194. DOI: 10.1109/VLHCC.2014.6883051.
- [83] Tony Lowe. “Debugging: The Key to Unlocking the Mind of a Novice Programmer?” In: *2019 IEEE Frontiers in Education Conference (FIE)*. Covington, KY, USA: IEEE Press, 2019, pp. 1–9. DOI: 10.1109/FIE43999.2019.9028699. URL: <https://doi.org/10.1109/FIE43999.2019.9028699>.

- [84] Stuart Garner. “A tool to support the use of part-complete solutions in the learning of programming”. In: *Proceeding de conférence*. 2001, pp. 222–228.
- [85] Jeroen J. G. Van Merriënboer and Marcel B. M. De Croock. “Strategies for Computer-Based Programming Instruction: Program Completion vs. Program Generation”. In: *Journal of Educational Computing Research* 8.3 (1992), pp. 365–394. DOI: 10.2190/MJDX-9PP4-KFMT-09PM. URL: <https://doi.org/10.2190/MJDX-9PP4-KFMT-09PM>.
- [86] Cornelia S. Große and Alexander Renkl. “Finding and fixing errors in worked examples: Can this foster learning outcomes?” In: *Learning and Instruction* 17.6 (2007), pp. 612–634. ISSN: 0959-4752. DOI: <https://doi.org/10.1016/j.learninstruc.2007.09.008>. URL: <https://www.sciencedirect.com/science/article/pii/S0959475207001004>.
- [87] Colleen M. Lewis, Ken Yasuhara, and Ruth E. Anderson. “Deciding to Major in Computer Science: A Grounded Theory of Students’ Self-Assessment of Ability”. In: *Proceedings of the Seventh International Workshop on Computing Education Research*. ICER ’11. Providence, Rhode Island, USA: Association for Computing Machinery, 2011, pp. 3–10. ISBN: 9781450308298. DOI: 10.1145/2016911.2016915. URL: <https://doi.org/10.1145/2016911.2016915>.
- [88] Hieke Keuning, Johan Jeuring, and Bastiaan Heeren. “A Systematic Literature Review of Automated Feedback Generation for Programming Exercises”. In: *ACM Trans. Comput. Educ.* 19.1 (Sept. 2018). DOI: 10.1145/3231711. URL: <https://doi.org/10.1145/3231711>.
- [89] Galina Deeva et al. “A review of automated feedback systems for learners: Classification framework, challenges and opportunities”. In: *Computers & Ed-*

- ucation 162 (2021), p. 104094. ISSN: 0360-1315. DOI: <https://doi.org/10.1016/j.compedu.2020.104094>. URL: <https://www.sciencedirect.com/science/article/pii/S036013152030292X>.
- [90] Jessica McBroom, Irena Koprinska, and Kalina Yacef. “A Survey of Automated Programming Hint Generation: The HINTS Framework”. In: *ACM Comput. Surv.* 54.8 (Oct. 2021). ISSN: 0360-0300. DOI: 10.1145/3469885. URL: <https://doi.org/10.1145/3469885>.
 - [91] Samiha Marwan et al. “Adaptive Immediate Feedback Can Improve Novice Programming Engagement and Intention to Persist in Computer Science”. In: ICER ’20. Virtual Event, New Zealand: Association for Computing Machinery, 2020, pp. 194–203. ISBN: 9781450370929. DOI: 10.1145/3372782.3406264. URL: <https://doi.org/10.1145/3372782.3406264>.
 - [92] Valerie J Shute. “Focus on formative feedback”. In: *Review of educational research* 78.1 (2008), pp. 153–189.
 - [93] Mary Catherine Scheeler, Kathy L Ruhl, and James K McAfee. “Providing performance feedback to teachers: A review”. In: *Teacher education and special education* 27.4 (2004), pp. 396–407.
 - [94] Sumit Gulwani, Ivan Radiček, and Florian Zuleger. “Automated clustering and program repair for introductory programming assignments”. In: *ACM SIGPLAN Notices* 53.4 (2018), pp. 465–480.
 - [95] Philip J Guo. “Online python tutor: embeddable web-based program visualization for cs education”. In: *Proceeding of the 44th ACM technical symposium on Computer science education*. 2013, pp. 579–584.

- [96] Ruan Reis et al. “Evaluating Feedback Tools in Introductory Programming Classes”. In: *2019 IEEE Frontiers in Education Conference (FIE)*. 2019, pp. 1–7. DOI: 10.1109/FIE43999.2019.9028418.
- [97] Alex Gerdes et al. “Ask-Elle: an adaptable programming tutor for Haskell giving automated feedback”. In: *International Journal of Artificial Intelligence in Education* 27.1 (2017), pp. 65–100.
- [98] Luciana Benotti et al. “The Effect of a Web-Based Coding Tool with Automatic Feedback on Students’ Performance and Perceptions”. In: *Proceedings of the 49th ACM Technical Symposium on Computer Science Education. SIGCSE ’18*. Baltimore, Maryland, USA: Association for Computing Machinery, 2018, pp. 2–7. ISBN: 9781450351034. DOI: 10.1145/3159450.3159579. URL: <https://doi.org/10.1145/3159450.3159579>.
- [99] Susanne Narciss. “Feedback strategies for interactive learning tasks”. In: *Handbook of research on educational communications and technology*. Routledge, 2008, pp. 125–143.
- [100] Samiha Marwan, Joseph Jay Williams, and Thomas Price. “An Evaluation of the Impact of Automated Programming Hints on Performance and Learning”. In: *Proceedings of the 2019 ACM Conference on International Computing Education Research. ICER ’19*. Toronto ON, Canada: Association for Computing Machinery, 2019, pp. 61–70. ISBN: 9781450361859. DOI: 10.1145/3291279.3339420. URL: <https://doi.org/10.1145/3291279.3339420>.
- [101] Antonija Mitrovic, Stellan Ohlsson, and Devon K. Barrow. “The effect of positive feedback in a constraint-based intelligent tutoring system”. In: *Computers & Education* 60.1 (2013), pp. 264–272. ISSN: 0360-1315. DOI: <https://doi.org/10.1016/j.compedu.2012.08.001>.

//doi.org/10.1016/j.compedu.2012.07.002. URL: <https://www.sciencedirect.com/science/article/pii/S0360131512001613>.

- [102] Ayelet Fishbach and Stacey R Finkelstein. “How feedback influences persistence, disengagement, and change in goal pursuit”. In: *Goal-directed behavior* (2012), pp. 203–230.

Appendix A

LIST OF ACTIVITIES - TUNEPAD.LIVE

Table 11.: List with Code Beats Activities 1

Begin of Table				
Day	Type	Song Name	Author	Link
1	Worked Example	Never Recover	Lil Baby, Gunna	https://tunepad.live/app/dropbook/52019
1	In Class	Knuck If You Buck	Crime Mob	https://tunepad.live/app/dropbook/51060
1	In Class	Mo Bamba	Sheck Wes	https://tunepad.live/app/dropbook/51066
1	After Class	Make No Sense	YoungBoy NBA	https://tunepad.live/app/dropbook/51067
2	Worked example	Bank Account	21 Savage	https://tunepad.live/app/dropbook/51510
2	In Class	EVERY CHANGE I GET	DJ Khaled	https://tunepad.live/app/dropbook/51071
2	In Class	Work REMIX	ASAP Ferg	https://tunepad.live/app/dropbook/51074
2	After Class	Bad Boy	Juice WRLD	https://tunepad.live/app/dropbook/51082
3	Worked example	God's Plan	Drake	https://tunepad.live/app/dropbook/51511
3	In Class	Solid (feat. Drake)	Young Stoner Life	https://tunepad.live/app/dropbook/51106
3	In Class	p r i d e . i s . t h e . d e v i l	J. Cole	https://tunepad.live/app/dropbook/51087
3	After Class	Nonstop	Drake	https://tunepad.live/app/dropbook/51108
4	Worked example	Next Episode	Dr. Dre	https://tunepad.live/app/dropbook/51781
4	In Class	Have Mercy	Cordae	https://tunepad.live/app/dropbook/51795
4	In Class	Russian Cream	Key Glock	https://tunepad.live/app/dropbook/51814
4	After Class	Straightenin	Migos	https://tunepad.live/app/dropbook/51847
5	Worked example	durag activity	Baby Keem	https://tunepad.live/app/dropbook/51856
5	In Class	Oppanese	Fredo Bang	https://tunepad.live/app/dropbook/51863
5	In Class	Tyler Herro	Jack Harlow	https://tunepad.live/app/dropbook/51865
5	After Class	Laugh Now Cry Later	Drake	https://tunepad.live/app/dropbook/51866
6	Worked example	Surf	Young Thug	https://tunepad.live/app/dropbook/51896
6	In Class	4 Quarters	MaxThaDemon	https://tunepad.live/app/dropbook/51898
6	In Class	Forever	Drake, Kanye, etc.	https://tunepad.live/app/dropbook/51900
6	After Class	Jumpman	Drake	https://tunepad.live/app/dropbook/51902
7	Worked example	Slippery	Migos, Gucci Mane	https://tunepad.live/app/dropbook/53216
7	In Class	Ball If I Want To	DaBaby	https://tunepad.live/app/dropbook/53401

Continuation of Table				
Day	Type	Song Name	Author	Link
7	In Class	Rockstar	DaBaby	https://tunepad.live/app/dropbook/53393
7	After Class	Freestyle	Lil Baby	https://tunepad.live/app/dropbook/53536
8	Worked example	Bout Me	Coi Leray	https://tunepad.live/app/dropbook/53538
8	In Class	Graduation	benny blanco, Juice WRLD	https://tunepad.live/app/dropbook/53543
8	In Class	Late At Night	Roddy Rich	https://tunepad.live/app/dropbook/53545
8	After Class	Red	\$NOT	https://tunepad.live/app/dropbook/53548
9	Worked example	Walk It Talk It	Migos ft. Drake	https://tunepad.live/app/dropbook/54308
9	In Class	Look Alive	Blocboy JB ft. Drake	https://tunepad.live/app/dropbook/54311
9	Sketch 1			https://tunepad.live/app/dropbook/54446
9	Sketch 2			https://tunepad.live/app/dropbook/54449
9	Sketch 3			https://tunepad.live/app/dropbook/54451
10	Worked example	ball w/o you	21 Savage	https://tunepad.live/app/dropbook/54736
10	In Class	What's Next	Drake	https://tunepad.live/app/dropbook/54737
End of Table				

Appendix B

LIST OF ACTIVITIES - TUNEPAD.COM

Table 12.: List with Code Beats Activities 2

Begin of Table				
Day	Type	Song Name	Author	Link
1	Worked example	Never Recover	Lil Baby, Gunna	https://tunepad.com/project/38097
1	Short Activity	Knuck If You Buck	Crime Mob	https://tunepad.com/project/38098
1	Short Activity	Mo Bamba	Sheck Wes	https://tunepad.com/project/38099
1	Long Activity	Make No Sense	YoungBoy NBA	https://tunepad.com/project/38100
2	Worked example	Bank Account	21 Savage	https://tunepad.com/project/38101
2	Short Activity	EVERY CHANGE I GET	DJ Khaled	https://tunepad.com/project/38102
2	Short Activity	Work REMIX	ASAP Ferg	https://tunepad.com/project/38103
2	Long Activity	Bad Boy	Juice WRLD	https://tunepad.com/project/38104
3	Worked example	God's Plan	Drake	https://tunepad.com/project/38106
3	Short Activity	Solid (feat. Drake)	Young Stoner Life	https://tunepad.com/project/38108
3	Short Activity	p r i d e . i s . t h e . d e v i l	J. Cole	https://tunepad.com/project/38109
3	Long Activity	Nonstop	Drake	https://tunepad.com/project/38110
4	Worked example	Next Episode	Dr. Dre	https://tunepad.com/project/38111
4	Short Activity	Have Mercy	Cordae	https://tunepad.com/project/38112
4	Short Activity	Russian Cream	Key Glock	https://tunepad.com/project/38113
4	Long Activity	Straightenin	Migos	https://tunepad.com/project/38114
5	Worked example	Oppanese	Fredo Bang	https://tunepad.com/project/38115
5	Short Activity	durag activity	Baby Keem	https://tunepad.com/project/38116
5	Short Activity	Tyler Herro	Jack Harlow	https://tunepad.com/project/38117
5	Long Activity	Laugh Now Cry Later	Drake	https://tunepad.com/project/38118
"6	Worked example	Surf	Young Thug	https://tunepad.com/project/38119
6	Short Activity	Graduation	benny blanco, Juice WRLD	https://tunepad.com/project/38120
6	Short Activity	4 Quarters	MaxThaDemon	https://tunepad.com/project/38121
6	Long Activity	Jumpman	Drake	https://tunepad.com/project/38122
7	Worked example	Slippery	Migos, Gucci Mane	https://tunepad.com/project/38123
7	Short Activity	Ball If I Want To	DaBaby	https://tunepad.com/project/38124

Continuation of Table				
Day	Type	Song Name	Author	Link
7	Short Activity	Rockstar	DaBaby	https://tunepad.com/project/38125
7	Long Activity	Freestyle	Lil Baby	https://tunepad.com/project/38126
8	Worked example	Bout Me	Coi Leray	https://tunepad.com/project/38127
8	Short Activity	Late At Night	Roddy Rich	https://tunepad.com/project/38128
8	Short Activity	Forever	Drake, Kanye, etc.	https://tunepad.com/project/38129
8	Long Activity	Red	\$NOT	https://tunepad.com/project/38130
End of Table				

Vita

Douglas Lusa Krug received his BSc. in Computer Science in 2007 from Centro Universitário de União da Vitória, Paraná, Brazil, and his MSc. in Applied Computing, concentration area in Computer Systems Engineering in 2018 from the Federal Technological University of Paraná - UTFPR, Paraná, Brazil. As a full-time graduate student in the Ph.D. in Computer Science program at Virginia Commonwealth University, his research focused entirely on developing and testing *Code Beats*, aligning his previous experience as a computer science professor, for more than four years, and his industry experience as a software engineer of more than twelve years.

Publications

- [1] Douglas Lusa Krug et al. “Inspiring Interest in Computing using Music: A Case Study on Students Lacking Prior Music Education”. In: *In Progress*. 2023.
- [2] Douglas Lusa Krug et al. “Scaffolding for Introductory Music-Based Programming Activities”. In: *In Progress*. 2023.
- [3] Douglas Lusa Krug et al. “Using Domain-Specific, Immediate Feedback to Support Students Learning Computer Programming to Make Music”. In: *Proceedings of the 2023 Conference on Innovation and Technology in Computer Science Education V. 1*. ITiCSE '23. Turku, Finland: Association for Computing Machinery, 2023. DOI: 10.1145/3587102.3588851. URL: <https://doi.org/10.1145/3587102.3588851>.
- [4] Douglas Lusa Krug et al. “Attracting Adults to Computer Programming via Hip Hop”. In: *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1*. SIGCSE 2023. Toronto ON, Canada: Association for Computing Machinery, 2023, pp. 528–534. ISBN: 9781450394314. DOI: 10.1145/3545945.3569800. URL: <https://doi.org/10.1145/3545945.3569800>.
- [5] Douglas Lusa Krug. “Code Beats - Teaching Computer Programming via Hip Hop Beats”. In: *Proceedings of the 27th ACM Conference on Innovation and Technology in Computer Science Education Vol. 2*. ITiCSE '22. Dublin, Ireland: Association for Computing Machinery, 2022, pp. 646–647.

ISBN: 9781450392006. DOI: 10.1145/3502717.3532111. URL: <https://doi.org/10.1145/3502717.3532111>.

- [6] Yifan Zhang et al. “A Case Study of Middle Schoolers’ Use of Computational Thinking Concepts and Practices during Coded Music Composition”. In: *Proceedings of the 27th ACM Conference on on Innovation and Technology in Computer Science Education Vol. 1*. ITiCSE ’22. Dublin, Ireland: Association for Computing Machinery, 2022, pp. 33–39. ISBN: 9781450392013. DOI: 10.1145/3502718.3524757. URL: <https://doi.org/10.1145/3502718.3524757>.
- [7] Douglas Lusa Krug et al. “Code Beats: A Virtual Camp for Middle Schoolers Coding Hip Hop”. In: *Proceedings of the 52nd ACM Technical Symposium on Computer Science Education*. SIGCSE ’21. Virtual Event, USA: Association for Computing Machinery, 2021, pp. 397–403. ISBN: 9781450380621. DOI: 10.1145/3408877.3432424. URL: <https://doi.org/10.1145/3408877.3432424>.